Android

网络开发从入门到精通

张 余 编著

涵盖

◎移动浏览器 ◎HTML 5开发客户端 ◎聊天系统 ◎Google API
◎RSS客户端 ◎Wi-Fi系统 ◎网络监控系统 ◎邮件系统

循序渐进、实例讲解：
基础知识、核心内容、100多个实例、一线开发人员的技术结晶



清华大学出版社

Android 网络开发从入门到精通

张　余　编　著

清华大学出版社
北　京

内 容 简 介

本书循序渐进地讲解了 Android 网络开发技术的基本知识，内容新颖、知识全面、讲解详细。全书共分 16 章，第 1～2 章是基础知识，讲解 Android 基础和 Android 网络开发基础；第 3 章详细讲解 HTTP 通信处理的基本知识；第 4 章详细讲解 URL 处理的实现过程；第 5 章详细讲解为 Android 开发网页的实现过程；第 6 章讲解 WebKit 浏览器的基本知识；第 7 章讲解在 Android 中开发蓝牙应用的基本知识；第 8 章讲解在 Android 中开发 Wi-Fi 应用的基本知识；第 9 章讲解在 Android 中开发 RSS 应用的基本知识；第 10 章讲解在 Android 中开发电子邮件应用的基本知识；第 11 章讲解让网络和多媒体接轨的基本知识；第 12 章讲解在 Android 中开发移动微博应用的基本知识；第 13 章讲解开发 Android 流量统计系统的基本知识；第 14 章讲解开发 Android 流量监控系统的基本知识；第 15 章和第 16 章是两个综合实例，分别讲解开发流量监控系统和电子邮件系统的基本流程。书中的每个实例都遵循先提出制作思路及包含知识点，再在实例最后补充总结知识点并引导读者举一反三。

本书定位于 Android 的初、中级用户，既可以作为初学者的参考书，也可以作为有一定 Android 基础读者的提高书。

图书在版编目(CIP)数据

Android 网络开发从入门到精通/张余编著. --北京：清华大学出版社，2014
ISBN 978-7-302-34192-5

Ⅰ. ①A…　Ⅱ. ①张…　Ⅲ. ①移动终端—应用程序—程序设计　Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字(2013)第 246342 号

责任编辑：李玉萍
装帧设计：杨玉兰
责任校对：王　晖
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn, 010-62791865
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　张：**38.25　　**字　数：**926 千字
版　　次：2014 年 1 月第 1 版　　**印　次：**2014 年 1 月第 1 次印刷
印　　数：1～3000
定　　价：75.00 元

产品编号：

前　　言

Foreword

进入 21 世纪以来，整个社会开始发生了深刻的变化。生活和工作的快节奏令我们目不暇接，各种各样的信息充斥着我们的视野、撞击着我们的思维。追忆过去，Windows 操作系统的诞生成就了微软的霸主地位，也造就了 PC 时代的繁荣。然而，以 Android 和 iPhone 手机为代表的智能移动设备的发明却敲响了 PC 时代的警钟！移动互联网时代已经来临，谁会成为这些移动设备的主宰？毫无疑问，它就是 Android——PC 时代的 Windows！

看 3G的璀璨绚丽

随着 3G 的到来，无线带宽的升高，使得更多内容丰富的应用程序安装在手机上成为可能，如视频通话、视频点播、移动互联网冲浪、在线看书/听歌、内容分享等。为了承载这些数据应用及快速部署，手机功能将会越来越智能，越来越开放。为了实现这些需求，必须有一个好的开发平台来支持，在此由 Google 公司发起的 OHA 联盟走在了业界的前列，于 2007 年 11 月推出了开放的 Android 平台，任何公司及个人都可以免费获取到源代码及开发 SDK。由于 Android 系统的开放性和优异性，Android 平台得到了业界广泛的支持，其中包括各大手机厂商和著名的移动运营商等。继 2008 年 9 月第一款基于 Android 平台的手机 G1 发布之后，预计三星、摩托罗拉、索爱、LG、华为等公司都将推出自 Gflg~Android 平台的手机，中国移动也联合各手机厂商共同推出基于 Android 平台的 OPhone。按目前的发展态势，我们有理由相信，Android 平台将在短时间内跻身智能手机开发平台前列。

自从 2009 年 3G 牌照在国内发放后，3G、Andriod、iPhone、Google、苹果、手机软件、移动开发等词不绝于耳。随着 3G 网络的大规模建设和智能手机的迅速普及，移动互联网时代已经微笑着迎面而来。

以创新的搜索引擎技术而一跃成为互联网巨头的 Google，无线搜索成为 Google 进军移动互联网的一块基石。早在 2007 年，Google 中国就把无线搜索当作战略重心，不断推出新产品，尝试通过户外媒体推广移动搜索产品，并积极与运营商、终端厂商、浏览器厂商等达成战略合作。

Android 操作系统是 Google 最具杀伤力的武器之一。苹果以其天才的创新，使得 iPhone 在全球迅速拥有了数百万忠实“粉丝”，而 Android 作为第一个完整、开放、免费的手机平台，使开发者在为其开发程序时拥有更大的自由。与 Windows Mobile、Symbian 等厂商不同的是，Android 操作系统免费向开发人员提供，这样可节省近三成成本，得到了众多厂商与开发者的拥护。自从 2011 年后，Android 就一直是市场占有率最高的智能手机系统，并且 Android 的成功也造就了使用 Android 系统的手机制造商。现在三星借助 Android 的东风，

成为世界上发货量最大的手机制造商。

巨大的优势

从技术角度而言，Android 与 iPhone 相似，采用 WebKit 浏览器引擎，具备触摸屏、高级图形显示和上网功能，用户可以在手机上查收电子邮件、搜索网址和观看视频节目等。Android 手机比 iPhone 等其他手机更强调搜索功能，界面更强大，可以说是一种融入了全部 Web 应用的平台。Android 的版本包括：Android 1.1、Android 1.5、Android 1.6、Android 2.0…当前的最新版本是 Android 4.2。随着版本的更新，从最初的触屏到现在的多点触摸，从普通的联系人到现在的数据同步，从简单的 GoogleMap 到现在的导航系统，从基本的网页浏览到现在的 HTML 5，这都说明 Android 已经逐渐稳定，而且功能越来越强大。此外，Android 平台不仅支持 Java、C、C++等主流编程语言，还支持 Ruby、Python 等脚本语言，甚至 Google 专为 Android 的应用开发推出了 Simple 语言，这使得 Android 有着非常广泛的开发群体。

移动网络的优越性

网络应用的最大优势就是移动互联网是开放的，因为封闭的平台(iPhone、iPad 等)让很多开发者转身离去，而投奔 Android、Windows Mobile 等平台。等到移动设备和桌面设备一样强大的时候，移动网络应用有可能和本地应用一决高下。

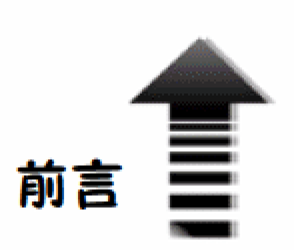
在现实情况下，有经验的网络开发者更有可能从事移动网络开发。开发者可以通过 HTML、CSS、JavaScript 等技术开发网络应用，而无须学习新的编程语言。对开发者来说，本地应用开发也许并不是最困难的事情，但网络开发将会是他们熟练掌握的技术。移动网络应用市场比本地应用市场更大。通过开发网络应用，开发者可以通过单一版本的应用获得广泛的用户群(来自各种平台的手机)。iPhone、Android 和 Windows Mobile 只代表了一小部分移动市场，而大部分手机都可以访问移动网络。

另外，移动互联网是一个开放平台，开发者无须等待长达数周的审查过程(结果却被拒之门外)。乔布斯无法像审查 iPhone 应用那样审查网络应用，而且开发者也无须等待。

本书的内容

本书循序渐进地讲解了开发 Android 网络应用的基本知识，并通过具体的实例讲解了各个知识点的具体用法。本书内容新颖、知识全面、讲解详细，全书共分 16 章，具体说明如下。

章　节	主要内容
第 1 章	Android 系统介绍
第 2 章	Android 网络开发基础
第 3 章	HTTP 通信处理
第 4 章	URL 处理
第 5 章	为 Android 开发网页



续表

章　节	主要内容
第 6 章	WebKit 浏览器详解
第 7 章	在 Android 中开发蓝牙应用
第 8 章	在 Android 中开发 Wi-Fi 应用
第 9 章	在 Android 中开发 RSS 应用
第 10 章	在 Android 中开发电子邮件应用
第 11 章	让网络和多媒体接轨
第 12 章	在 Android 中开发移动微博应用
第 13 章	Android 流量统计系统
第 14 章	开发流量监控系统
第 15 章	Android 网络典型应用实践
第 16 章	开发一个邮件系统

科学的学习方法

不要认为学习计算机网络是一件很困难的事情，不断寻找规律，学习新知识和新技能，积累经验，这几乎是每一个电脑高手的成长之路。中国有句古话：“授人以鱼，不如授人以渔。”说的是传授给人既有知识，不如传授给人学习知识的方法。通过本书，我们将告诉读者学习的方法，并介绍一条比较清晰的学习之路。

1) 积极的心态

无论是知识还是技能，智者之所以能够更好更快地掌握这些知识和技能，很大程度上得益于良好的学习方法。人们常说：兴趣是最好的老师，压力是前进的动力，要想获得一个积极的心态，最好能对学习对象保持浓厚的兴趣。如果暂时提不起兴趣，那么试试把来自工作或生活的压力，转化为学习的动力。

2) 注重实践

建议读者在学习本书的过程中，学完理论后，进行实际操作。首先学习书中的理论，再动手调试本书中的实例，然后用模拟器运行书中的例子。只有这样才能做到印象深刻，真正理解 Android 网络的基本知识。这样当在实际应用中遇到其他类似问题时，才能做到熟能生巧、触类旁通。

3) 学以致用

对于计算机网络技术，除了少部分专业人士外，大部分人学习网络的目的是为了应用，通过网络解决工作中的问题并提高工作效率。“解决问题”常常是促使人们学习的一大动机，带着问题学习，不但进步快，而且很容易对网络产生更大的兴趣，从而获得持续的进步。

(1) 利用网络资源和课外资料

在学习过程中，难免会遇到自己不理解的知识，此时可以找一些相关的书籍来阅读，

不断尝试解决问题。或者通过互联网的搜索引擎找到问题的解决办法，善用搜索引擎，基本上可以找到大多数问题的所在！

(2) QQ 群

如果在互联网找不到问题的解决办法，可以通过 QQ 访问相关学习群，群中的网络高手们会对你提出的问题进行回答。

(3) 向网络高手学习

在练习实际操作能力时，可以虚心向网络领域的高手学习。如果读者闭门造车，盲人摸象，则很难掌握技术精髓。而经过身边网络高手的指点，可以轻松掌握相关的技能。

本书的特色

本书内容相当丰富，并且覆盖全面，可以满足 Android 网络开发技术人员成长道路上的方方面面。我们的目标是通过一本图书，提供多本图书的价值，读者可以根据自己的需要有选择地阅读，以完善本人的知识和技能结构。在内容的编写上，本书具有以下特色。

1. 结构合理

从用户的实际需要出发，科学安排知识结构，内容由浅入深，叙述清楚，并附有相应的总结和练习，具有很强的知识性和实用性，反映了当前 Android 网络开发技术的发展和应用水平。同时全书精心筛选了最具代表性、读者最关心的典型知识点，这些知识点几乎包括了 Android 网络开发技术的所有方面。

2. 易学易懂

本书条理清晰、语言简洁，可帮助读者快速掌握每个知识点；每个部分既相互连贯又自成体系，使读者既可以按照本书编排的章节顺序进行学习，也可以根据自己的需求对某一章节进行针对性的学习。

3. 实用性强

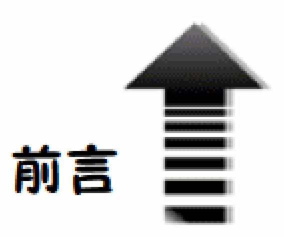
本书彻底摒弃枯燥的理论和简单的操作，注重实用性和可操作性，将 Android 网络开发技术的理论融合到实际的操作环境中，使用户在掌握相关操作技能的同时，还能够学习到相应的开发知识。

4. 实例典型

书中的开发实例不仅典型而且具有创意，将传统互联网的内容/服务与移动平台紧密结合起来，体现了移动互联网应用所需的创新精神及良好的用户体验理念，这个设计思路非常值得大家去思考和学习。

本书的读者对象

本书在内容安排上由浅入深，写作上层层剥洋葱式的分解，实例充分举证，非常适合于入门 Android 网络开发技术的初学者，同时也适合于具有一定 Android 开发基础，想对



Android 网络开发技术进一步了解和掌握的中级学者。如果你是以下类型的学者，此书会带领你迅速进入 Android 开发领域。

- 有一定 Android 开发经验的读者。
- 从事 Android 开发的研究人员和工作人员。
- 有一定的 Android 基础，想快速学会 Android 高级技术的读者。
- 有一定 Android 开发基础，需要加深对 Android 技术核心进一步了解和掌握的程序员。
- 高等院校相关专业的学生，或需要编写论文的学生。
- 企业和公司在职人员、需要提高学习或工作需要的程序员。
- 从事 Android 移动网络开发等相关工作的技术人员。

在本书的写作过程中得到了清华大学出版社工作人员的大力支持，在此特意感谢各位编辑老师们的指点和付出的汗水。另外，笔者本人毕竟水平有限，如有纰漏和不尽如人意之处在所难免，诚请读者提出意见或建议，以便修订并使之更臻完善。

本书部分源代码网络下载路径：http://www.tup.tsinghua.edu.cn。

编　者



目　录

Contents

第 1 章　Android 系统介绍 1

1.1　Android 是一款智能手机 2
1.1.1　什么是智能手机 2
1.1.2　当前主流的智能手机系统 2
1.2　Android 的巨大优势 4
1.3　搭建 Android 开发环境 5
1.3.1　安装 Android 系统的要求 5
1.3.2　安装 Android 插件 6
1.3.3　设定 Android SDK 主目录 16
1.4　创建 Android 虚拟设备 17
1.4.1　Android 模拟器简介 18
1.4.2　模拟器和真机的区别 18
1.4.3　创建 Android 虚拟设备 19
1.4.4　启动 AVD 模拟器 20
1.4.5　快速安装 SDK 的方法 21
1.5　搭建环境过程中的常见问题 22

第 2 章　Android 网络开发基础 27

2.1　Android 安装文件介绍 28
2.1.1　Android SDK 目录结构 28
2.1.2　android.jar 及内部结构 29
2.1.3　SDK 帮助文档 30
2.1.4　解析 Android SDK 实例 32
2.2　分析 Android 的系统架构 32
2.2.1　Android 体系结构介绍 32
2.2.2　Android 工程文件结构 35
2.2.3　应用程序的生命周期 38
2.3　网页开发基础 41
2.3.1　HTML 简介 41
2.3.2　XML 技术 42
2.3.3　CSS 技术 44
2.3.4　JavaScript 技术 45
2.4　简析 Android 内核 45
2.4.1　Android 继承于 Linux 45
2.4.2　Android 内核和 Linux 内核的区别 46
2.5　简要分析 Android 源码 48
2.5.1　获取并编译 Android 源码 48
2.5.2　Android 对 Linux 的改造 50
2.5.3　为 Android 构建 Linux 的操作系统 50
2.6　总结和网络应用有关的包 51

第 3 章　HTTP 通信处理 53

3.1　Java 中的网络通信基础 54
3.1.1　Java 网络通信概述 54
3.1.2　Socket 和 ServerSocket 55
3.1.3　网络通信的综合应用 59
3.2　HTTP 协议 63
3.2.1　HTTP 概述 63
3.2.2　协议功能 64
3.2.3　Android 中的 HTTP 65
3.3　使用 Apache 接口 66
3.3.1　Apache 接口基础 66
3.3.2　Apache 应用基础 66
3.3.3　Apache 应用要点 73
3.4　使用标准 Java 接口 82
3.4.1　IP 地址 83
3.4.2　套接字 Socket 类 84
3.5　使用 Android 网络接口 85
3.5.1　android.net.http 中的类 85
3.5.2　在手机屏幕中传递 HTTP 参数 85

第 4 章 URL 处理......91

4.1 使用 URL 类......92
4.1.1 URL 类基础......92
4.1.2 URI 和 URL 的使用......95
4.2 使用 URLConnection 类......103
4.3 使用 HttpURLConnection 类......111
4.3.1 HttpURLConnection 的主要用法......111
4.3.2 在 Android 中使用 HttpURLConnection 类......114

第 5 章 为 Android 开发网页......121

5.1 准备工作......122
5.1.1 搭建开发环境......122
5.1.2 简单网页开发......125
5.1.3 控制页面的缩放......128
5.2 为 Android 中的网页添加 CSS 样式......129
5.2.1 编写基本样式......129
5.2.2 添加视觉效果......131
5.3 为 Android 网页添加 JavaScript 特效......133
5.3.1 jQuery 框架介绍......133
5.3.2 具体实践......134
5.4 在 Android 网页中使用 Ajax 特效......137
5.5 让 Android 网页充满灵动活力......143
5.5.1 开源框架——JQTouch......143
5.5.2 JQTouch 简单应用......143
5.6 为网页增加数据存储功能......152
5.6.1 在 Android 网页中使用 Web Storage......152
5.6.2 在 Android 网页中使用 Web SQL Database......156

第 6 章 WebKit 浏览器详解......167

6.1 WebKit 的目录结构......168
6.2 WebKit 框架介绍......169
6.2.1 Java 层框架......170
6.2.2 C 层框架......172
6.3 WebKit 操作......174
6.3.1 WebKit 初始化......175
6.3.2 载入数据......176
6.3.3 刷新绘制......177
6.4 WebView 类详解......178
6.4.1 WebView 概述......178
6.4.2 实现 WebView 的两种方式......180
6.4.3 WebView 的常见功能......183
6.4.4 使用 WebView 类浏览网页......190
6.4.5 使用 WebView 类加载 HTML 程序......193
6.4.6 使用 WebView 加载 JavaScript 程序......194
6.4.7 使用 WebView 的注意事项......198

第 7 章 在 Android 中开发蓝牙应用......199

7.1 蓝牙系统的结构......200
7.1.1 蓝牙概述......200
7.1.2 蓝牙层次结构......201
7.1.3 蓝牙在 Android 和 Linux 中的差异......203
7.2 分析蓝牙源码......204
7.2.1 初始化蓝牙芯片......204
7.2.2 蓝牙服务......204
7.2.3 管理蓝牙电源......205
7.3 和蓝牙相关的类......206
7.3.1 BluetoothSocket 类......206
7.3.2 BluetoothServerSocket 类......207
7.3.3 BluetoothAdapter 类......208
7.3.4 BluetoothClass.Service 类......215
7.3.5 BluetoothClass.Device.Major 类......215
7.3.6 BluetoothClass.Device 类......216
7.3.7 BluetoothClass 类......216
7.4 Android 蓝牙的基本应用......218
7.4.1 使用 BluetoothAdapter 类......218
7.4.2 使用 BluetoothSocket 类......221
7.4.3 在 Android 平台开发蓝牙应用的基本步骤......223

7.5 开发一个遥控器——蓝牙控制玩具车......229

第 8 章 在 Android 中开发 Wi-Fi 应用......239

8.1 了解 Wi-Fi 系统的结构......240
8.1.1 Wi-Fi 概述......240
8.1.2 Wi-Fi 层次结构......240
8.1.3 Wi-Fi 在 Android 和 Linux 中的差异......242
8.2 分析 Wi-Fi 源码......242
8.2.1 本地部分......243
8.2.2 JNI 部分......246
8.2.3 Java FrameWork 部分......248
8.2.4 Setting 中的设置部分......249
8.3 开发 Wi-Fi 应用程序......250
8.3.1 WifiManager 类......250
8.3.2 在 Android 系统中控制 Wi-Fi......254
8.3.3 在 Android 系统中打开或关闭 Wi-Fi 网卡......262

第 9 章 在 Android 中开发 RSS 应用......267

9.1 RSS 基础......268
9.1.1 RSS 的用途......268
9.1.2 RSS 阅读器......268
9.1.3 RSS 的语法......269
9.2 SAX 介绍......270
9.2.1 SAX 的原理......270
9.2.2 基于对象和基于事件的接口......271
9.2.3 常用的接口和类......272
9.3 开发一个 RSS 订阅程序......275
9.3.1 实现界面布局文件......276
9.3.2 实现主程序文件......278
9.4 开发一个 RSS 阅读器......290
9.4.1 建立实体类......290
9.4.2 主程序文件 ActivityMain.java......293
9.4.3 实现 ContentHandler......296
9.4.4 主程序文件 ActivityShowDescription.java......299
9.4.5 主布局文件 main.xml......300
9.4.6 详情布局文件 showdescription.xml......300

第 10 章 在 Android 中开发电子邮件应用......305

10.1 使用 Android 的内置邮件系统......306
10.1.1 Android 邮件客户端配置......306
10.1.2 调用内置邮件系统在发送短信时实现 E-mail 通知......309
10.1.3 调用内置邮件系统在来电时实现自动邮件通知......314
10.1.4 调用内置邮件系统实现邮件发送......316
10.1.5 调用内置 Gmail 发送邮件......321
10.1.6 其他方法......325
10.2 使用 SmsManager 收发邮件......327
10.2.1 SmsManager 基础......327
10.2.2 使用 SmsManager 发送短信......329
10.2.3 解决 Android 邮件附件中文名乱码问题......335
10.3 使用包 commons-mail.jar 和 mail.jar......335
10.3.1 使用 commons-mail.jar 发送邮件......335
10.3.2 使用 mail.jar 接收邮件......339
10.3.3 Android 中用 commons-email.jar 和 mail.jar 收发邮件......345

第 11 章 让网络和多媒体接轨......347

11.1 MediaPlayer 视频技术详解......348
11.1.1 MediaPlayer 基础......348
11.1.2 MediaPlayer 的状态......348
11.1.3 MediaPlayer 方法的有效状态和无效状态......351

11.1.4 MediaPlayer 的接口............353
11.1.5 MediaPlayer 的常量............353
11.1.6 MediaPlayer 的公共方法......354
11.2 VideoView 技术详解......................355
11.2.1 VideoView 的构造函数........355
11.2.2 VideoView 的公共方法........356
11.3 在 Android 中播放网络上的 MP3...357
11.4 在 Android 中下载在线铃声..........365
11.5 在 Android 中上传文件到远程服务器..371
11.6 在 Android 中开发一个远程下载系统..375
11.6.1 基础知识介绍......................375
11.6.2 具体实现..............................378
11.7 在 Android 中开发一个网络视频播放器..383
11.7.1 实现布局文件......................384
11.7.2 实现显示文本值文件..........385
11.7.3 主程序文件..........................385
11.8 在 Android 中开发一个网络收音机..393
11.8.1 基本思路..............................393
11.8.2 演示代码..............................393

第 12 章 在 Android 中开发移动微博应用..397

12.1 微博介绍..398
12.2 微博开发技术介绍..........................399
12.2.1 XML-RPC 技术...................399
12.2.2 Meta Weblog API 客户端.....401
12.3 在 Android 上开发移动博客发布器..401
12.3.1 XML 请求............................402
12.3.2 常用接口..............................402
12.3.3 具体实现..............................403
12.4 详解腾讯 Android 版微博 API........410
12.4.1 源码和 jar 包下载................410
12.4.2 具体使用..............................411
12.5 详解新浪 Android 版微博 API........415
12.5.1 新浪微博图片缩放的开发实例....................................417
12.5.2 添加分享到新浪微博...........423
12.5.3 通过 Json 对象登录新浪微博....................................428
12.5.4 实现 OAuth 认证..................430

第 13 章 流量统计系统..........................433

13.1 流量统计基础................................434
13.1.1 TrafficStats 类......................434
13.1.2 Android 流量统计的基本思路....................................435
13.1.3 读取 Linux 内核获取流量信息....................................438
13.2 基于防火墙的流量统计..................444
13.3 适用 Android 系统的通用流量统计函数......................................447

第 14 章 流量监控系统..........................453

14.1 实现流量监控功能的方式..............454
14.2 系统需求分析................................454
14.3 系统需求.......................................455
14.4 编写布局文件................................455
14.4.1 主界面布局文件 main.xml.....456
14.4.2 帮助界面布局文件 help_dialog.xml.....................458
14.5 编写主程序文件............................458
14.5.1 实现服务勾选处理和模式设置功能..............................458
14.5.2 实现帮助模块......................470
14.5.3 实现公共库函数..................471
14.5.4 实现广播模块......................482
14.5.5 删除针对软件的设置规则....483
14.5.6 登录验证.............................483
14.5.7 打开或关闭某一个实施控件....................................484
14.6 系统测试.......................................486

第 15 章 Android 网络典型应用实践....489

15.1 测试网络下载速度..........................490

15.2 通过 Handler 实现异步消息处理 494
15.2.1 实现 HTTP 通信和 XML 解析的演示 495
15.2.2 使用 Handler 实现异步消息处理 501
15.3 实现网络多线程断点下载 506
15.3.1 实现原理 506
15.3.2 具体实现 506
15.4 判断当前网络 GPRS 和 Wi-Fi 的状态 519
15.4.1 ConnectivityManager 类和 NetworkInfo 类 520
15.4.2 在程序启动时对网络状态进行判断 522
15.5 开启或关闭 APN 523
第 16 章 开发一个邮件系统 527
16.1 项目介绍 528
16.1.1 项目背景 528
16.1.2 项目目的 528
16.2 系统需求分析 529
16.2.1 构成模块 529
16.2.2 系统流程 532
16.2.3 功能结构图 533
16.2.4 系统功能说明 533
16.2.5 系统需求 534
16.3 数据存储设计 535
16.3.1 用户信息类 535
16.3.2 SharedPreferences 539
16.4 具体编码 542
16.4.1 欢迎界面 542
16.4.2 系统主界面 545
16.4.3 邮箱类型设置 553
16.4.4 邮箱收取设置 556
16.4.5 邮箱发送设置 562
16.4.6 邮箱用户检查 567
16.4.7 设置用户别名 573
16.4.8 用户邮件编辑 576
16.5 打包、签名和发布 586
16.5.1 申请会员 586
16.5.2 生成签名文件 589
16.5.3 使用签名文件 595
16.5.4 发布 597

Android

第1章 Android系统介绍

Android 是 2007 年推出的一款智能手机平台，它是建立在 Linux 的开源基础之上的，能够迅速建立手机软件的解决方案。虽然 Android 的外形比较简单，但是其功能却十分强大，已经成为一个新兴的热点，并且成为市场占有率排名第一的智能手机操作系统。本章将简单介绍 Android 系统的相关知识，让读者了解 Android 的发展之路。

1.1 Android 是一款智能手机

其实在 Android 系统诞生之前，智能手机就已经大大丰富了人们的生活，受到广大手机用户的追捧。各大手机厂商在利益的驱动之下，纷纷建立了自己的智能手机操作系统来抢夺市场份额。Android 系统就是在这个风起云涌的历史背景下诞生的。

1.1.1 什么是智能手机

智能手机是指具有像个人电脑那样强大的功能，拥有独立的操作系统，用户可以自行安装游戏等第三方服务商提供的程序，并且可以通过移动通信网络来接入无线网络。在 Android 系统诞生之前，市面上已经有多款智能手机产品，例如 Symbian 和微软的 Windows Mobile 系列等。

一般来说，智能手机必须具备下面几个标准功能。

(1) 操作系统必须支持新应用的安装。

(2) 高速度处理芯片。

(3) 支持播放式的手机电视。

(4) 大存储芯片和存储扩展能力。

(5) 支持 GPS 导航。

根据上述标准，手机联盟公布了智能手机的主要特点。

(1) 应具备普通手机的所有功能。例如，可以进行正常的通话和收发短信等基本的手机应用。

(2) 是一个开放性的操作系统，在系统上可以安装更多的应用程序，从而实现功能的无限扩充。

(3) 具备上网功能。

(4) 具备 PDA 的功能，能够实现个人信息管理、日程记事、任务安排、多媒体应用、浏览网页等功能。

(5) 可以根据个人需要扩展机器的功能。

(6) 扩展性能强，并且可以支持众多第三方软件。

1.1.2 当前主流的智能手机系统

目前市面上最主流的智能手机系统当属 Windows Mobile、塞班 Symbian、Palm、黑莓 BlackBerry、苹果 iOS 和本书的主角 Android。

1. 微软的Windows Mobile

Windows Mobile 是微软公司的一款杰出产品，它将熟悉的 Windows 桌面扩展到了个人设备中。使用 Windows Mobile 操作系统的设备主要有 PPC 手机、PDA、随身音乐播放器等。Windows Mobile 操作系统有三种，分别是 Windows Mobile Standard、Windows



Mobile Professional 和 Windows Mobile Classic。

2. 塞班Symbian

Symbian 系统出自由诺基亚、索尼爱立信、摩托罗拉、西门子等几家大型移动通信设备商共同出资组建的一个合资公司，专门研发手机操作系统。现已被诺基亚全额收购。Symbian 有着良好的界面，采用内核与界面分离技术，对硬件的要求比较低，支持 C++、Visual Basic 和 J2ME。目前根据人机界面的不同，Symbian 体系的 UI(User Interface，用户界面)平台分为 Series60、Series80、Series90、UIQ 等。其中，Series60 主要是给数字键盘手机用；Series80 是为完整键盘所设计；Series90 则是为触控笔方式而设计。

注意： ① 2010 年 9 月，诺基亚宣布将从 2011 年 4 月起从 Symbian 基金会(Symbian Foundation)手中收回 Symbian 操作系统控制权。由此看来，诺基亚在 2008 年全资收购塞班公司之后希望继续扩大塞班影响力的愿望并没有实现。

② 在苹果和 Android 的强大市场攻势下，诺基亚在 2011 年 2 月 11 日宣布与微软达成广泛战略合作关系，并将 Windows Phone 作为其主要的智能手机操作系统。这家芬兰手机巨头试图通过结盟扭转颓势。截止本书成稿时，诺基亚和微软联合推出了最新版本 Windows Phone 8。

③ 2011 年 8 月 15 日，谷歌和摩托罗拉移动公司共同宣布，谷歌将以每股 40.00 美元现金收购摩托罗拉移动，总额约 125 亿美元，相比摩托罗拉移动股份的收盘价溢价了 63%，双方董事会都已全票通过该交易。谷歌 CEO 拉里 • 佩奇表示，摩托罗拉移动将完全专注于 Android 系统，收购摩托罗拉移动之后，将增强整个 Android 生态系统。佩奇同时表示，Android 将继续开源，收购的一个目的是为了获得专利。

3. Palm

Palm 是流行的个人数字助理(PDA，又称掌上电脑)的传统名字。从广义上讲，Palm 是 PDA 的一种，是 Palm 公司发明的。而从狭义上讲，Palm 是 Palm 公司生产的 PDA 产品，区别于 Sony 公司的 Clie 和 Handspring 公司的 Visor/Treo 等其他运行 Palm 操作系统的 PDA 产品。其显著特点之一是写入装置输入数据的方法，能够点击显示器上的图标选择输入的项目。2009 年 2 月 11 日，Palm 公司 CEO Ed Colligan 宣布以后将专注于 WebOS 和 Windows Mobile 的智能设备，而将不会再有基于“Palm OS”的智能设备推出，除了 Palm Centro 会在以后和其他运营商合作时继续推出。

4. 黑莓BlackBerry

BlackBerry 是加拿大 RIM 公司推出的一种移动电子邮件系统终端，其特色是支持推动式电子邮件、手机、文字短信、互联网传真、网页浏览及其他无线资讯服务，其最大优势是收发邮件。正因为这一优势，所以受到了商务用户的格外青睐。

5. iOS

iOS 作为苹果移动设备 iPhone 和 iPad 的操作系统，在 App Store 的推动之下，成为世界上引领潮流的操作系统之一。原本这个系统名为“iPhone OS”，直到 2010 年 6 月 7 日

WWDC 大会上宣布改名为“iOS”。iOS 推出的理念是能够使用多点触控屏幕的方式来操控手机。控制方法包括滑动、轻触开关及按键。与系统交互包括滑动(Swiping)、轻按(Tapping)、挤压(Pinching，通常用于缩小)及反向挤压(Reverse Pinching or unpinching，通常用于放大)。此外通过其自带的加速器，可以令其旋转设备改变其 y 轴以使屏幕改变方向，这样的设计令 iPhone 更便于使用。

从最初的 iPhone OS 演变至最新的 iOS 系统，它横跨 iPod Touch、iPad、iPhone，成为苹果最强大的操作系统，甚至新一代的 Mac OS X Lion 也借鉴了 iOS 系统的一些设计。可以说 iOS 是苹果的又一个成功的操作系统，能给用户带来极佳的使用体验。

6. Android

Android 是我们本书的主角，是谷歌于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统的名称。Android 平台由操作系统、中间件、用户界面和应用软件组成，号称是首个为移动终端打造的真正开放和完整的移动软件。

1.2 Android 的巨大优势

从 2007 年 11 月 5 日诞生之日起，到 2011 年 7 月，Android 系统在智能手机的占有率高达 43%，位居智能手机系统占有率排行榜的第一位。并且随着各大厂商新产品的推出，必然会继续巩固这一地位。为什么 Android 能在这么多的智能系统中脱颖而出，成为市场占有率第一的手机系统呢？要想分析其原因，需要先了解它的巨大优势，分析究竟是哪些优点吸引了厂商和消费者的青睐。

1. 系出名门

Android 出身于 Linux 家族，是一款号称开源的手机操作系统。当 Android“一炮走红”之后，各大手机联盟纷纷加入，并且都推出了各自系列产品。这个联盟由包括中国移动、三星、摩托罗拉、高通、宏达电子和 T-Mobile 等在内的 30 多家技术和无线应用的领军企业组成。通过与运营商、设备制造商、开发商和其他有关各方结成深层次的合作伙伴关系，希望借助建立标准化、开放式的移动电话软件平台，在移动产业内形成一个开放式的生态系统。

2. 开发团队的支持

Android 的研发队伍阵容豪华，包括摩托罗拉、Google、HTC(宏达电子)、Philips、T-Mobile、高通、魅族、三星、LG 以及中国移动在内的 34 家企业，他们都将基于该平台开发手机的新型业务，应用之间的通用性和互联性将在最大程度上得到保持。

3. 诱人的奖励机制

谷歌为了提高程序员的开发积极性，不但为他们提供了一流的硬件设置和软件服务，而且还采取了振奋人心的奖励机制，定期举行比赛，创意和应用夺魁者将会得到重奖。

4. 开源

开源意味着对开发人员和手机厂商来说，Android 是完全无偿免费使用的。因为源代码公开的原因，所以激发了全世界各地无数程序员的热情。于是很多手机厂商纷纷采用 Android 作为自己产品的系统，甚至包括很多山寨厂商。而对于开发人员来说，众多厂商的采用就意味着人才需求大，所以纷纷加入到 Android 开发大军中来。于是有一些干的还可以的程序员经不住高薪诱惑，都纷纷改行做 Android 开发。至于“混”得不尽如人意的程序员，就更加坚定了“改行做 Android 手机开发”，目的是想寻找自己程序员生涯的转机。并且有很多遇到发展瓶颈的程序员，也决定做 Android 开发，因为这样可以学习一门新技术，使自己的未来更加有保障。

1.3　搭建 Android 开发环境

“工欲善其事，必先利其器”出自《论语》，意思是要想高效地完成一件事，需要有一个合适的工具。对于 Android 开发人员来说，开发工具同样至关重要。作为一项新兴技术，在进行开发前首先要搭建一个对应的开发环境。而在搭建开发环境前，需要了解安装开发工具所需要的硬件和软件配置条件。

在具体搭建 Android 开发环境之前，首先需要明确 Android 开发包括底层开发和应用开发两部分。

- 底层开发：大多数是指和硬件相关的开发，并且是基于 Linux 环境的，例如开发驱动程序。
- 应用开发：是指开发能在 Android 系统上运行的程序，例如游戏、地图等程序。本书的重点是讲解多媒体应用开发，即使讲一些底层的知识，也是为上层的应用服务的。

另外，因为目前市面上最主流的操作系统是 Windows，所以本书只介绍在 Windows 环境下搭建 Android 开发环境的过程。

1.3.1　安装Android系统的要求

在搭建开发环境之前，一定先确定基于 Android 应用软件所需要开发环境的要求，具体如表 1-1 所示。

表 1-1　开发系统要求

项　目	版本要求	说　明	备　注
操作系统	Windows XP/Windows 7 /Windows 8	根据自己的电脑自行选择	选择自己最熟悉的操作系统
软件开发包	Android SDK	选择最新版本的 SDK	截止到目前，最新手机版本是 4.1，最普及的版本是 2.3

续表

项　目	版本要求	说　明	备　注
IDE	Eclipse IDE+ADT	Eclipse 3.3 以上版本和 ADT(Android Development Tools)开发插件	选择 for Java Developer
其他	JDK Apache Ant	Java SE Development Kit 5 或 6	不能选择单独的 JRE 进行安装，必须要有 JDK

Android 开发工具是由多个开发包组成的，其中最主要的开发包如下。

- JDK：可以到网址 http://www.oracle.com/technetwork/java/javase/downloads/index.html 下载。
- Eclipse：可以到网址 http://www.eclipse.org/downloads/下载 Eclipse IDE for Java Developers。
- Android SDK：可以到网址 http://developer.android.com 下载。
- 下载对应的开发插件。

1.3.2　安装Android插件

本书的安装是以 Windows 7 为平台，安装的软件为 JDK 1.6、Eclipse 3.3、ADT 1.5、Android SDK 2.3。下面具体介绍各自的安装步骤，并且在配套的视频中有详细的介绍。

1．安装JDK

安装 Eclipse 的开发环境需要 JRE 的支持，在 Windows 上安装 JRE/JDK 非常简单，流程如下。

(1) 在 Sun 官方网站下载，网址为 http://www.oracle.com/technetwork/java/javase/downloads/index.html，如图 1-1 所示。

(2) 在图 1-1 中可以看到有很多版本，运行 Eclipse 时虽然只需要 JRE 就可以了，但是在开发 Android 应用程序的时候，却需要完整的 JDK(JDK 已经包含了 JRE)，并且要求其版本在 1.5 以上，这里选择 Java SE (JDK) 6。其下载页面如图 1-2 所示。

(3) 在图 1-2 中找到“JDK 6 Update 22”项目，单击其右侧的 Download 按钮后弹出填写登录信息界面，在此输入你的账号信息，如果没有账号可以免费注册一个。然后单击 Continue 按钮，如图 1-3 所示。

(4) 进入“选择操作系统和语言”界面，在此首先选择 Windows 选项，然后单击 Download 按钮，如图 1-4 所示。

(5) 下载安装文件 jdk-6u22-windows-i586.exe。

(6) 下载完成后双击 jdk-6u22-windows-i586.exe 开始安装，将弹出安装向导对话框。然后单击“下一步”按钮，如图 1-5 所示。

(7) 弹出“自定义安装”界面，在其中选择文件的安装路径，如图 1-6 所示。

(8) 单击“下一步”按钮，开始安装，如图 1-7 所示。





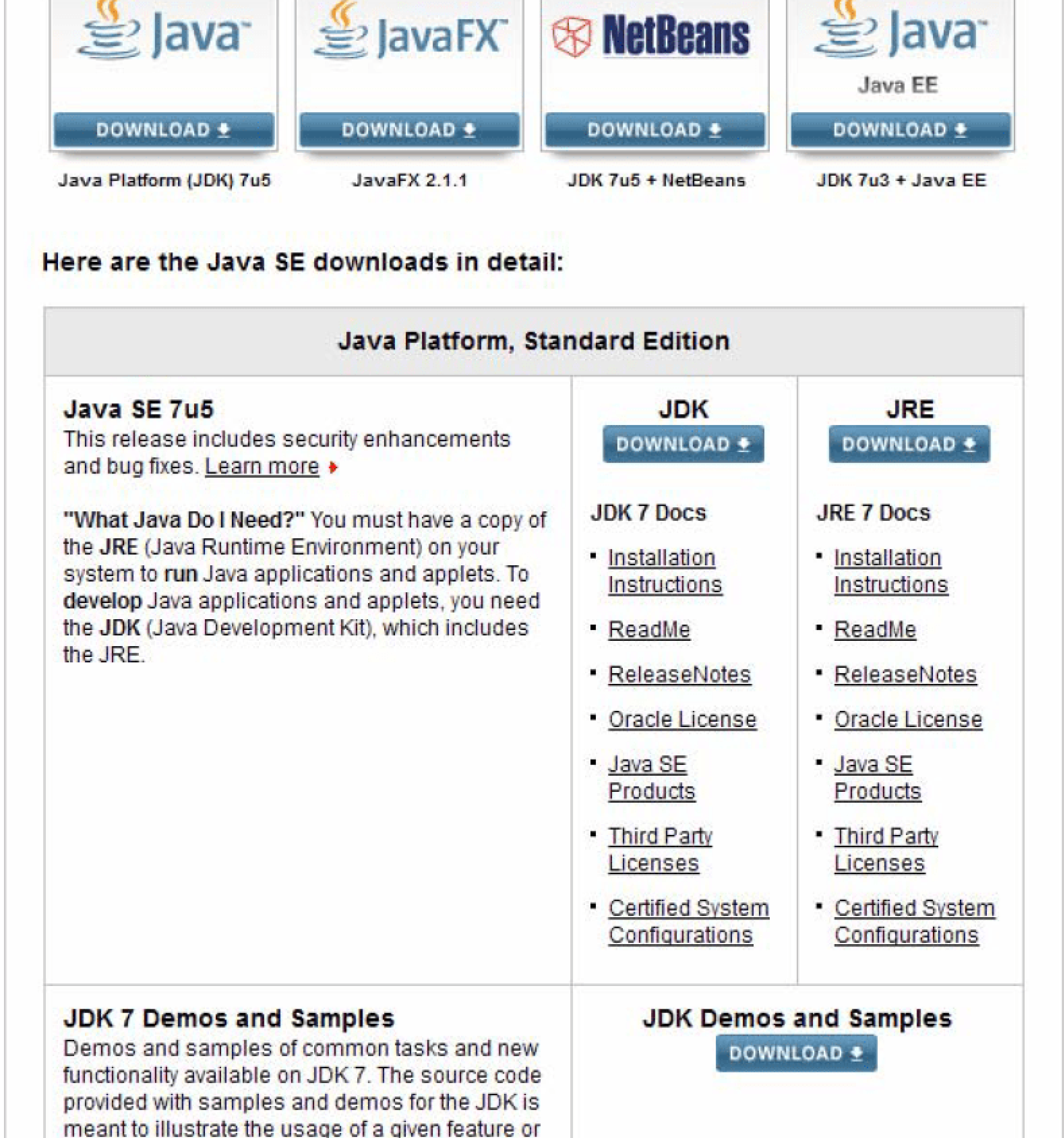


图 1-1　Sun官方下载页面

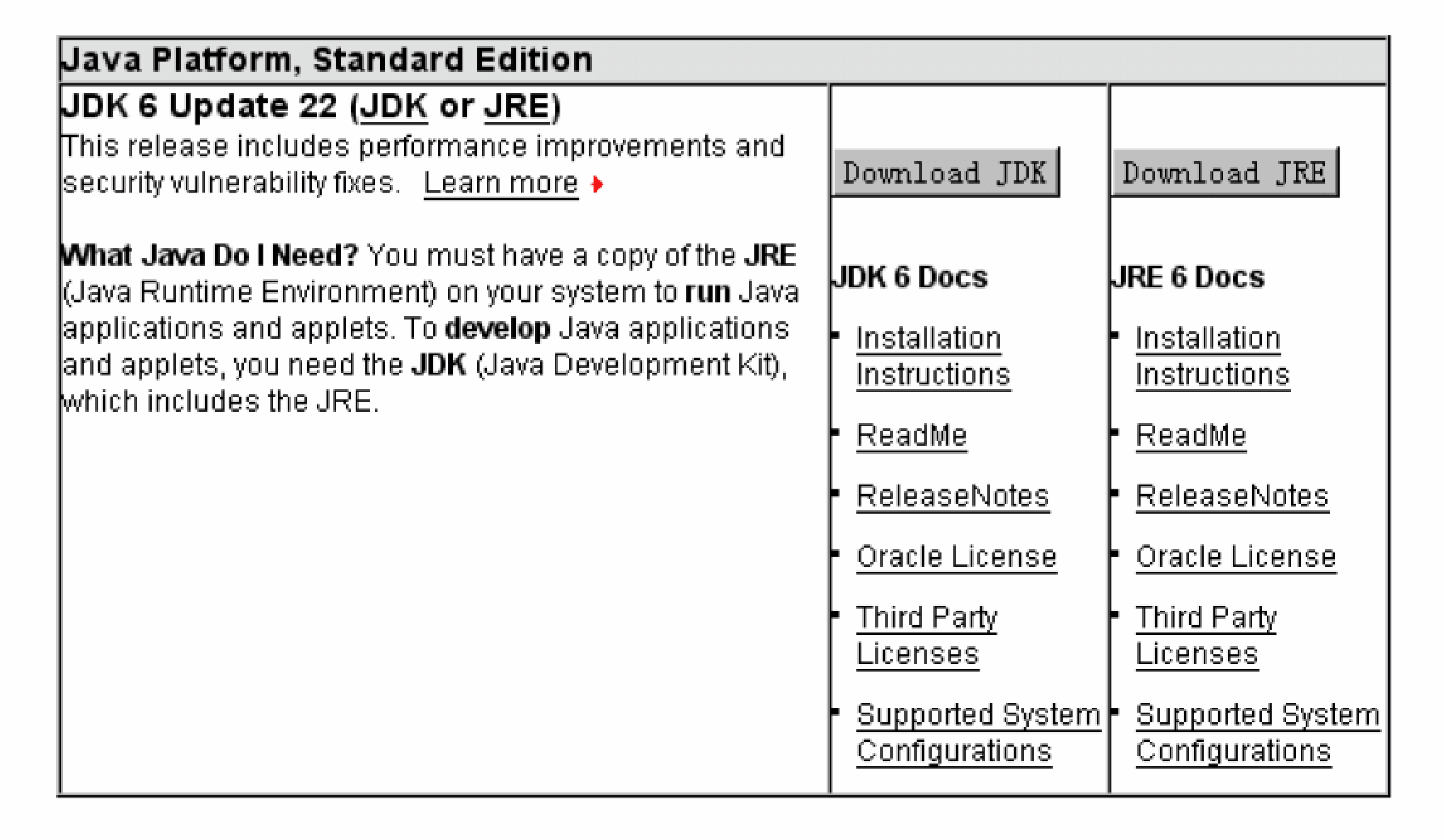


图 1-2　JDK下载页面

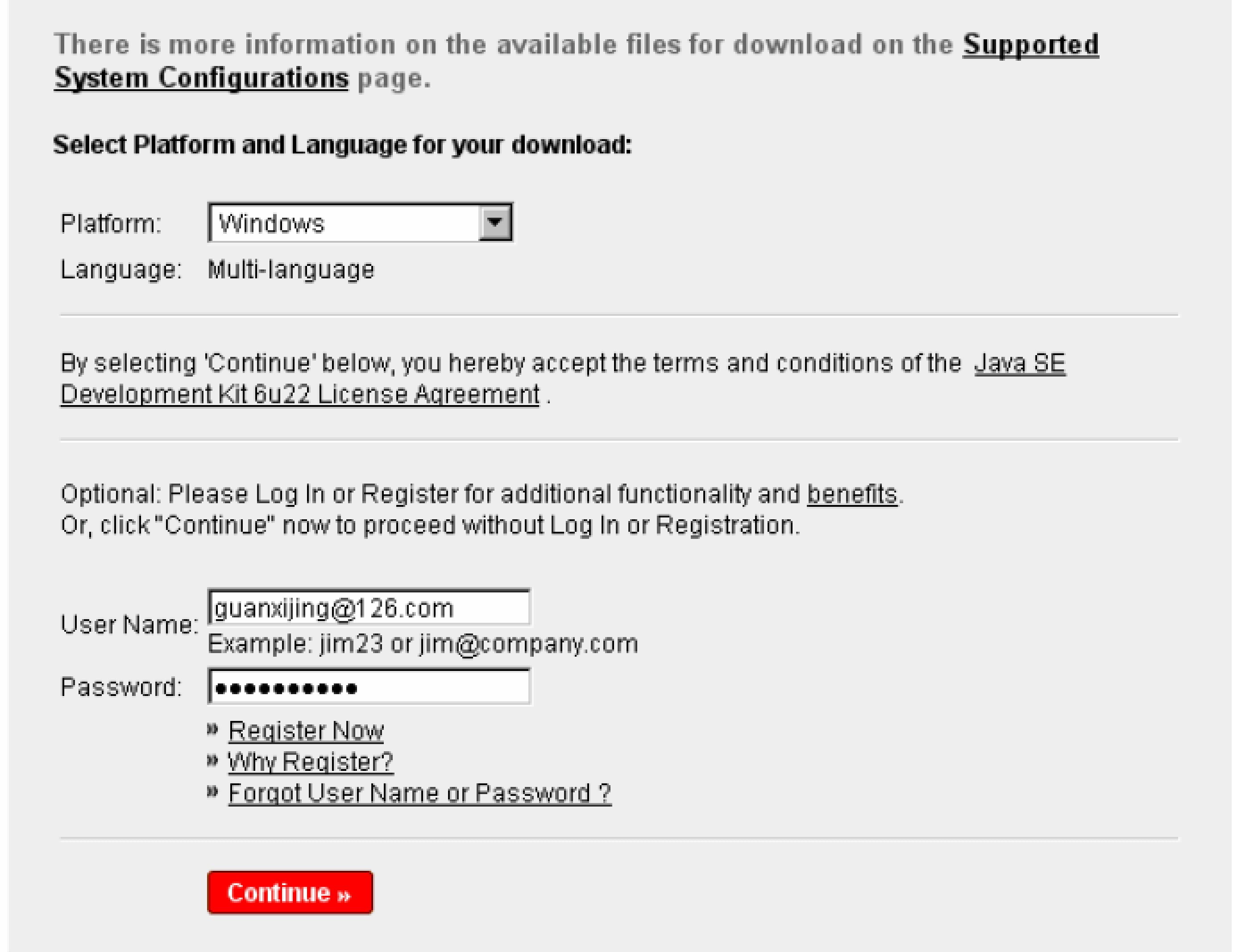


图 1-3　输入账号信息

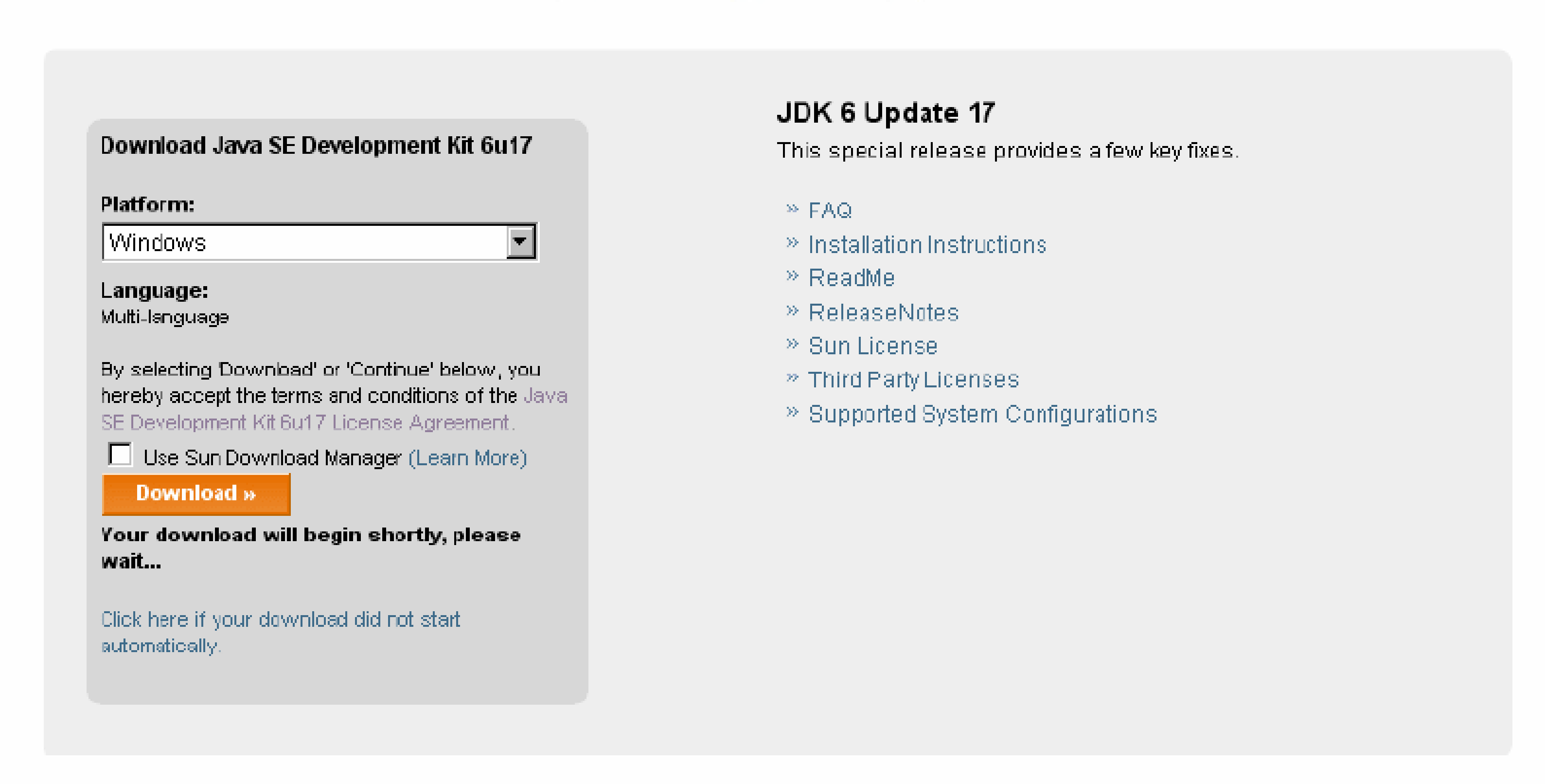


图 1-4　选择Windows选项



图 1-5　安装向导对话框



图 1-6 “自定义安装”界面

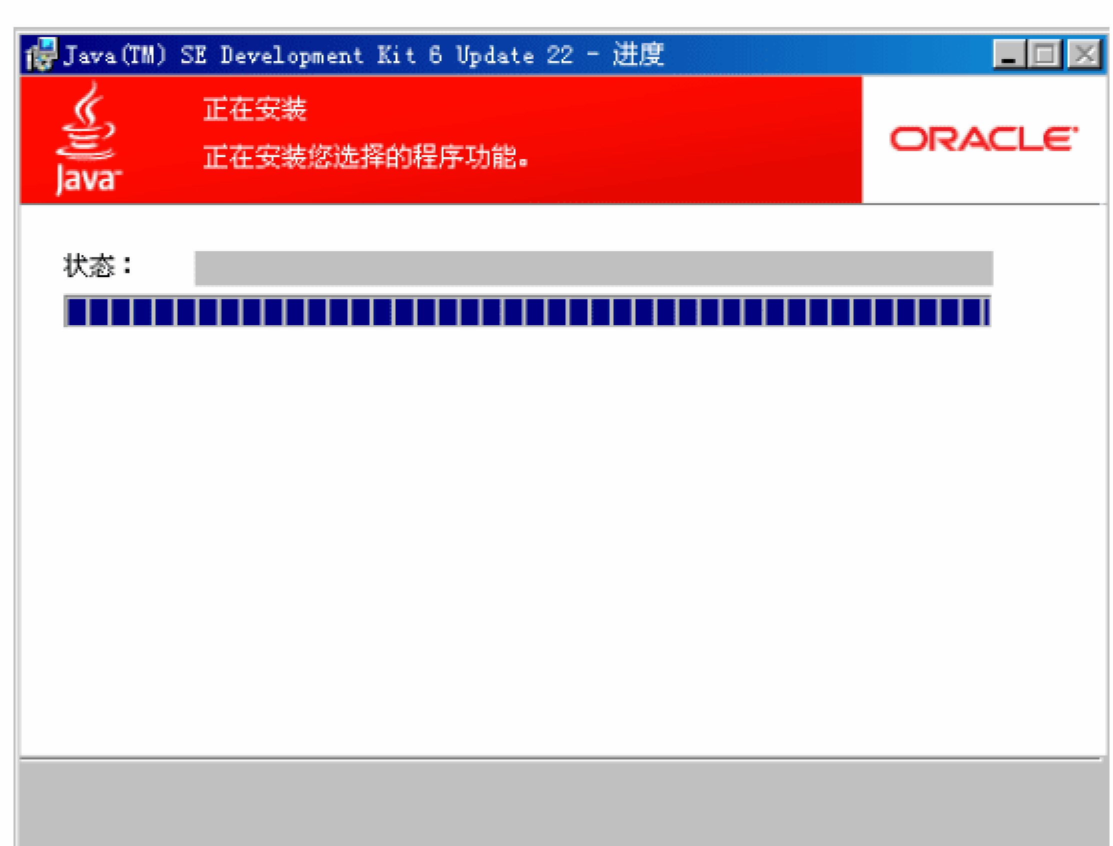


图 1-7 开始安装

(9) 弹出“目标文件夹”界面，在其中选择要安装的位置，如图 1-8 所示。

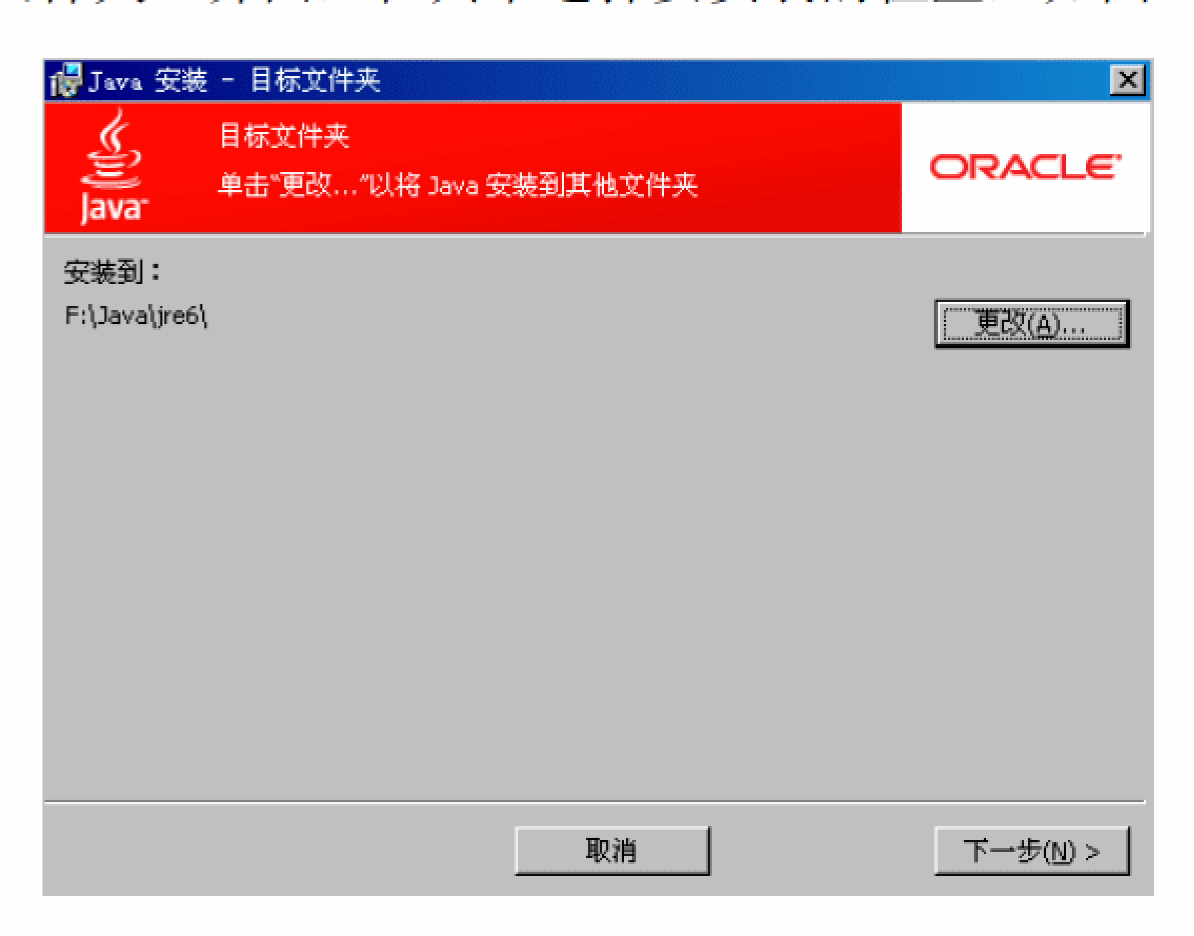


图 1-8 “目标文件夹”界面

(10) 单击“下一步”按钮后继续安装，如图 1-9 所示。



图 1-9 继续安装

(11) 弹出已成功安装界面，单击“完成”按钮完成整个安装过程，如图 1-10 所示。



图 1-10　完成安装

注意：完成安装后可以检测是否安装成功，方法是：依次选择“开始”|“运行”命令，在“运行”对话框中输入“cmd”并按 Enter 键，在打开的 CMD 窗口中输入“java-version”，如果显示如图 1-11 所示的提示信息，则说明安装成功。

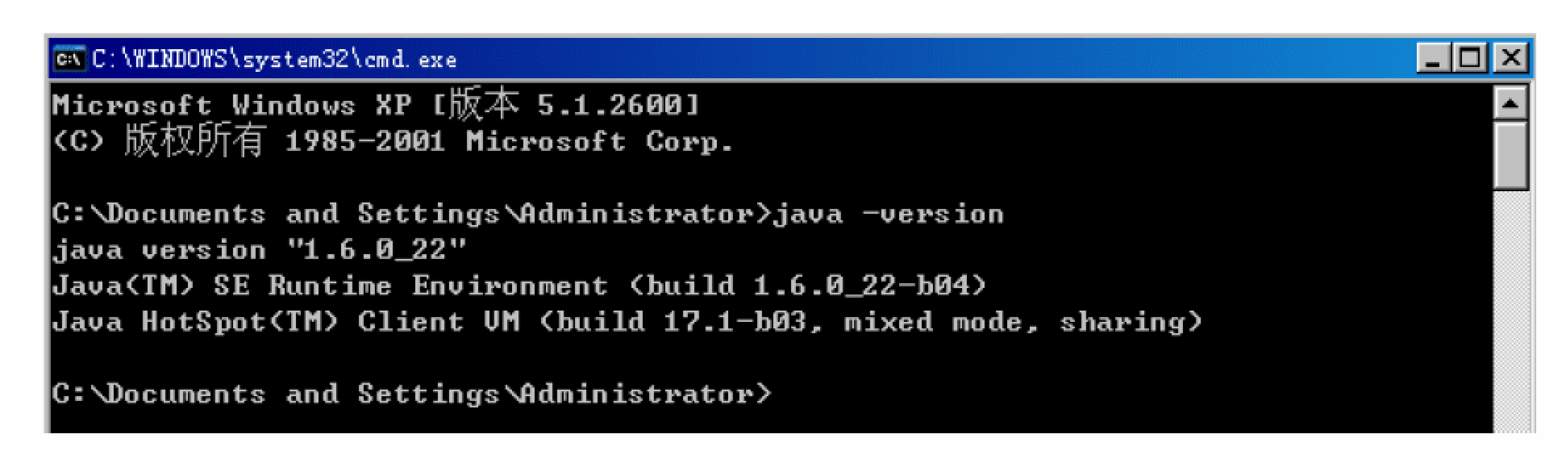


图 1-11　CMD窗口

如果检测到没有安装成功，则需要将其目录的绝对路径添加到系统的 PATH 中。具体做法如下。

(1) 右击“我的电脑”，依次选择“属性”|“高级”命令，再单击“环境变量”，然后在“系统变量”处选择新建、在“变量名”处输入“JAVA_HOME”、在“变量值”中输入刚才的目录，比如笔者的目录是 C:\Program Files\Java\jdk1.6.0_22，如图 1-12 所示。

(2) 再次新建一个变量名 classpath，其变量值如下：

```
.;%JAVA_HOME%/lib/rt.jar;%JAVA_HOME%/lib/tools.jar
```

单击“确定”按钮找到 PATH 的变量，双击或单击编辑，在变量值最前面添加如下值：

```
%JAVA_HOME%/bin;
```

具体如图 1-13 所示。

(3) 依次选择“开始”|“运行”命令，在“运行”对话框中输入“cmd”并按 Enter 键，在打开的 CMD 窗口中输入“java-version”，如果显示如图 1-14 所示的提示信息，则说明安装成功。

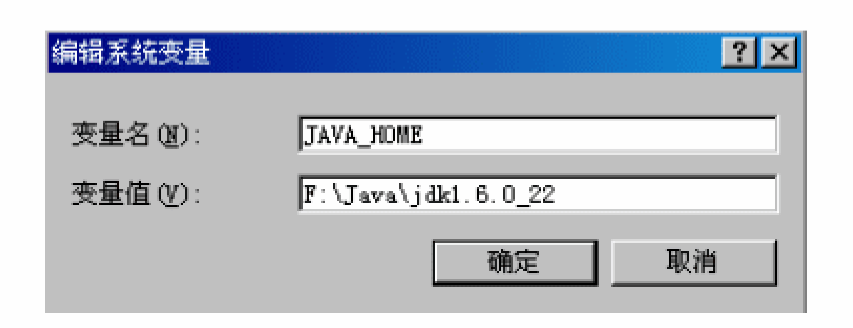


图 1-12　新建变量“JAVA_HOME”

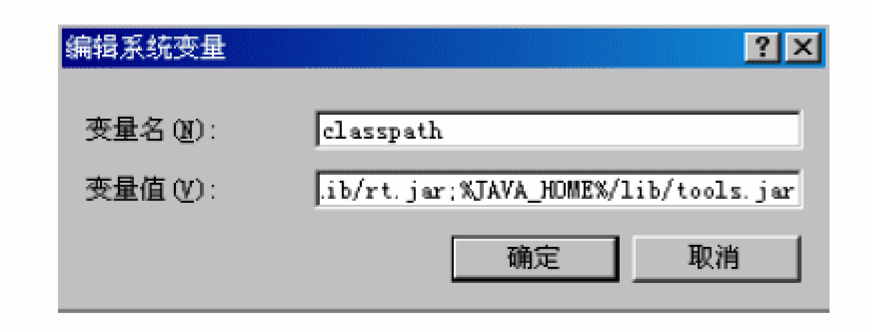


图 1-13　新建变量“classpath”

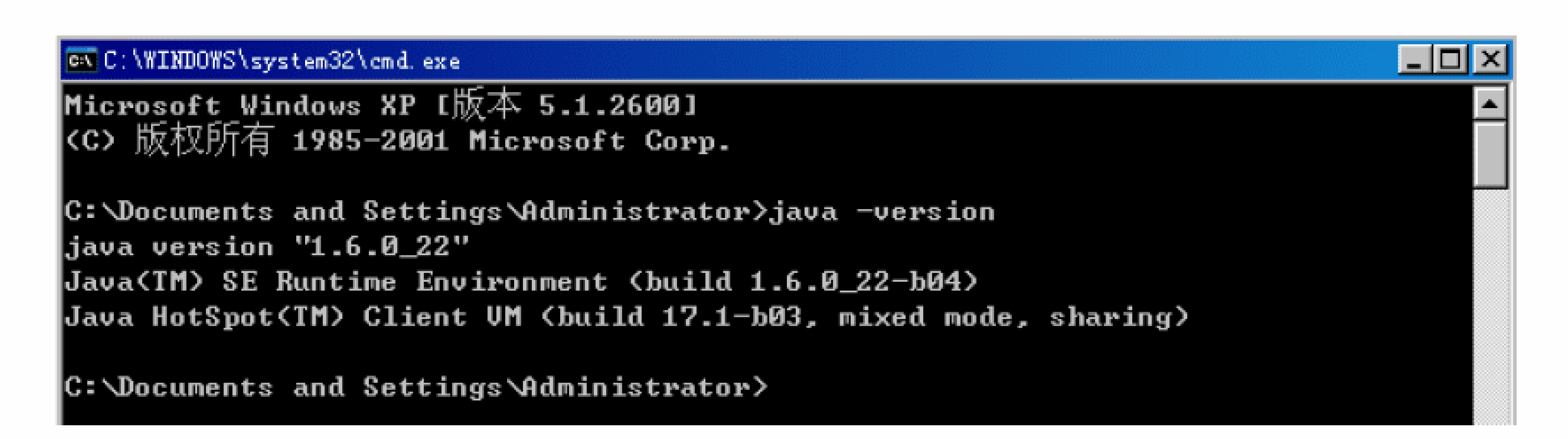


图 1-14　CMD界面

注意：上述变量是按照笔者本人的安装路径设置的，笔者安装 JDK 的路径是 C:\Program Files\Java\jdk1.6.0_22。

2. 安装Eclipse

在安装好 JDK 后，就可以开始安装 Eclipse 了，具体安装步骤如下。

(1) 打开 Eclipse 的官方下载页面 http://www.eclipse.org/downloads/，如图 1-15 所示。

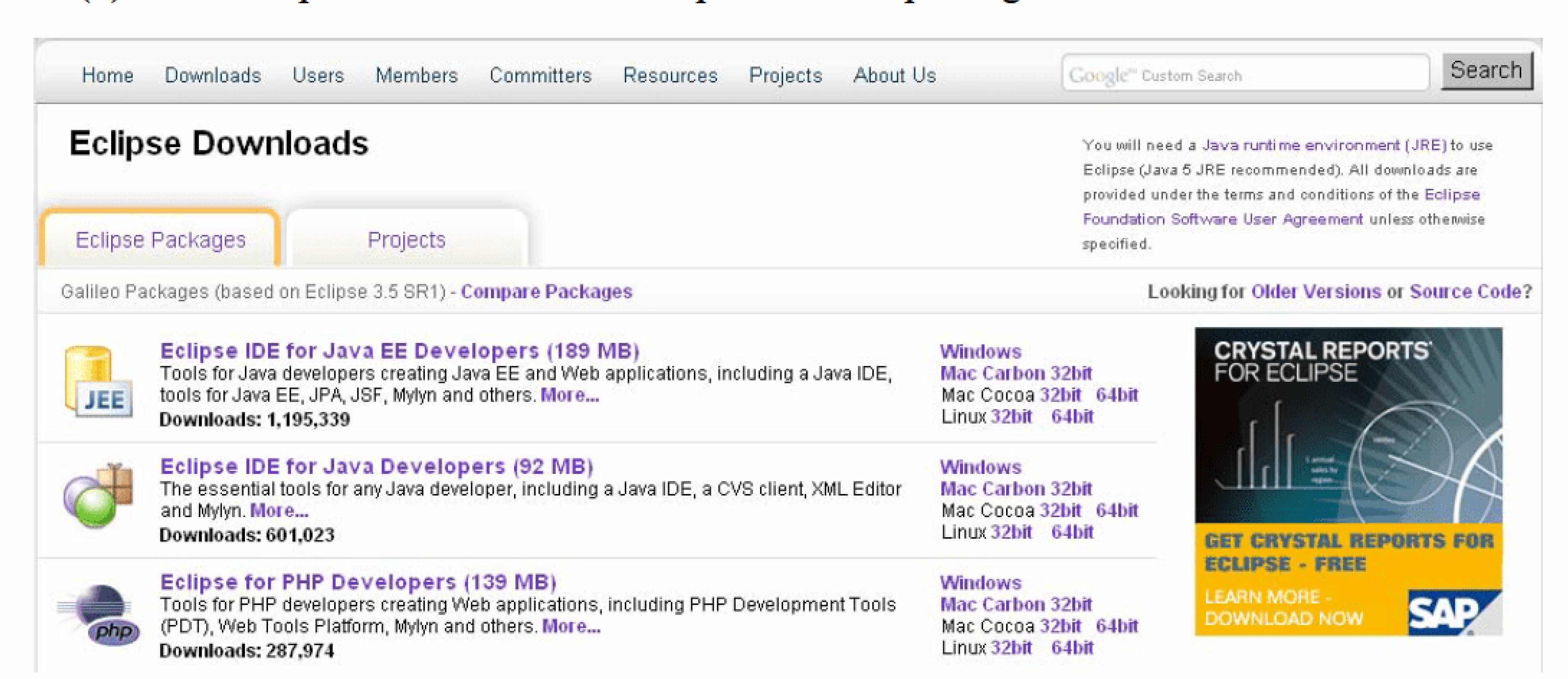


图 1-15　下载页面

(2) 在图 1-15 所示的界面中选择 Eclipse IDE for Java Developers (92 MB)选项，进入其下载的镜像页面，在此只需选择离用户最近的镜像即可(一般推荐的下载速度就可以)，如图 1-16 所示。

(3) 下载完成后，先找到下载的压缩包 eclipse-java-galileo-SR1-win32.zip。

注意：解压下载的 Eclipse 压缩文件后就可以使用，而无须执行安装程序，不过在使用前一定要先安装 JDK。在此假设 Eclipse 解压后存放的目录为 F:\eclipse。

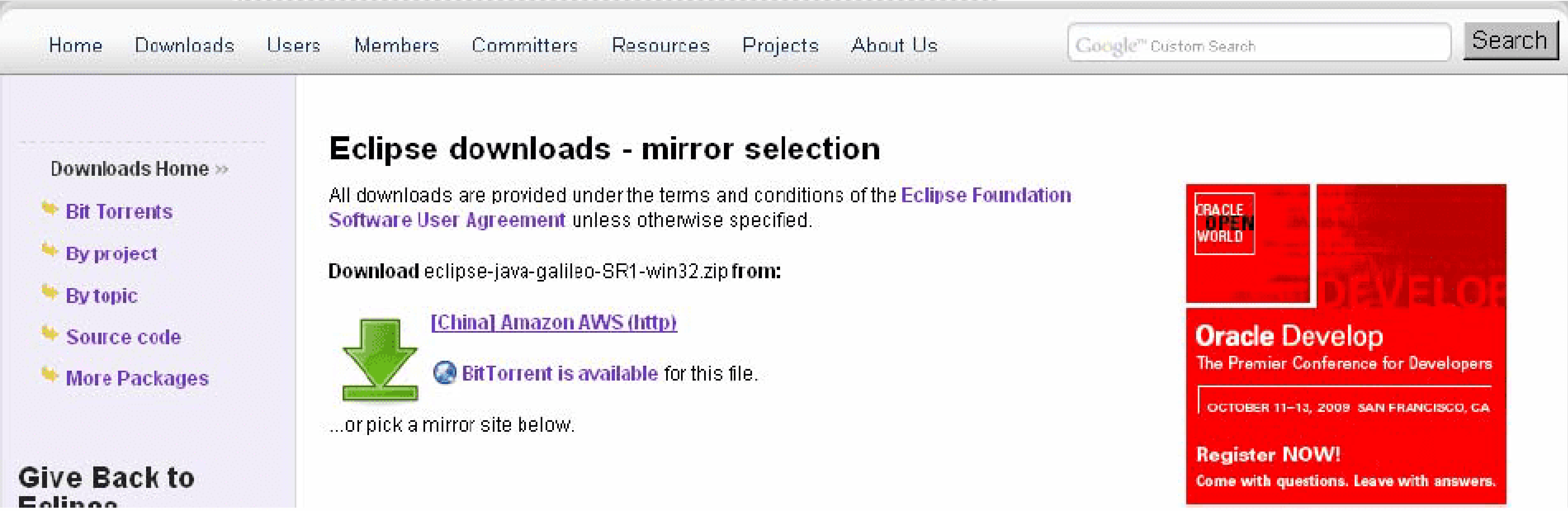


图 1-16　选择镜像

(4) 进入解压后的目录，此时可以看到一个名为“eclipse.exe”的可执行文件，双击此文件直接运行，Eclipse 能自动找到用户先前安装的 JDK 路径。启动界面如图 1-17 所示。

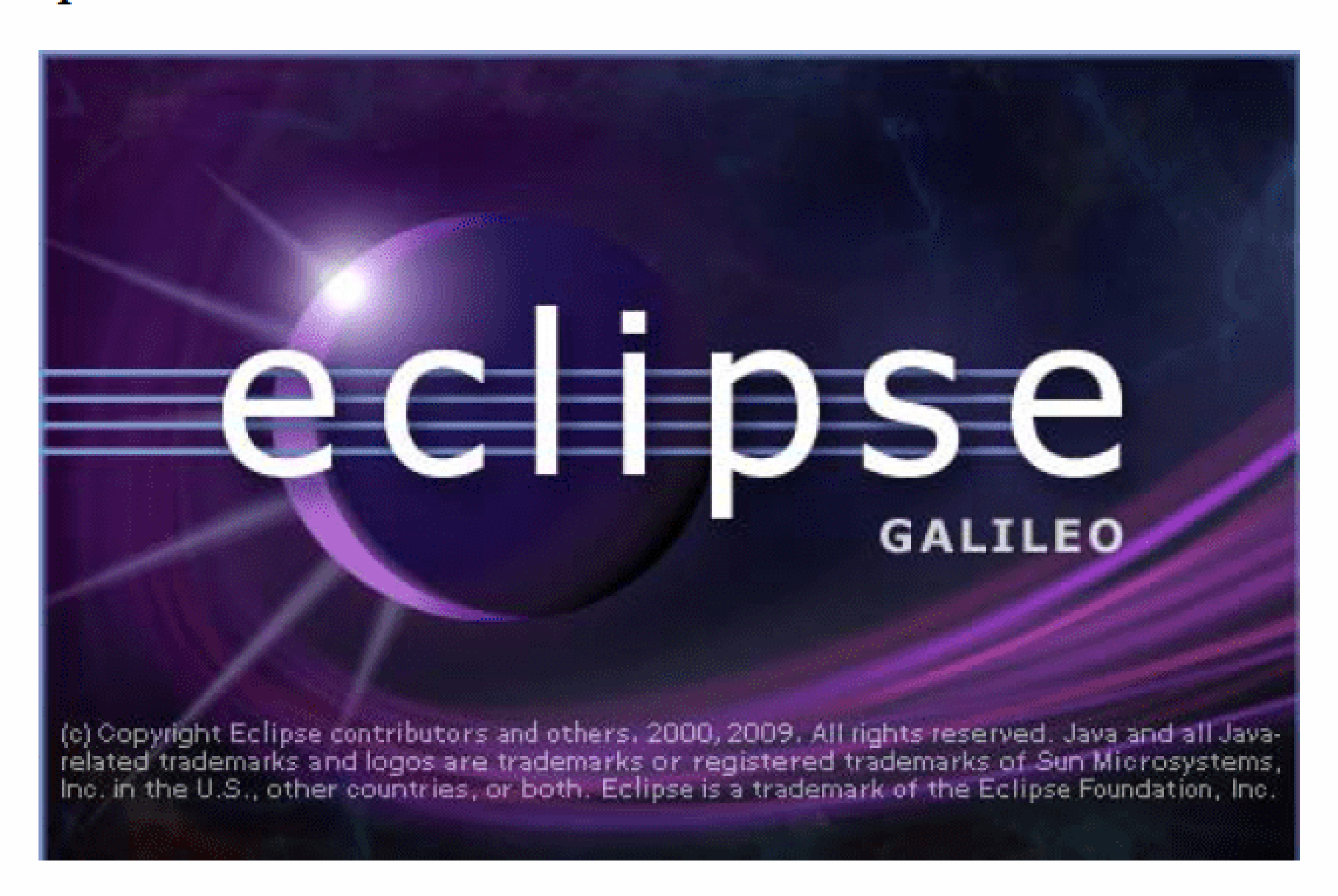


图 1-17　Eclipse启动界面

因为是安装后第一次启动 Eclipse，所以会看到选择工作空间的提示，如图 1-18 所示。

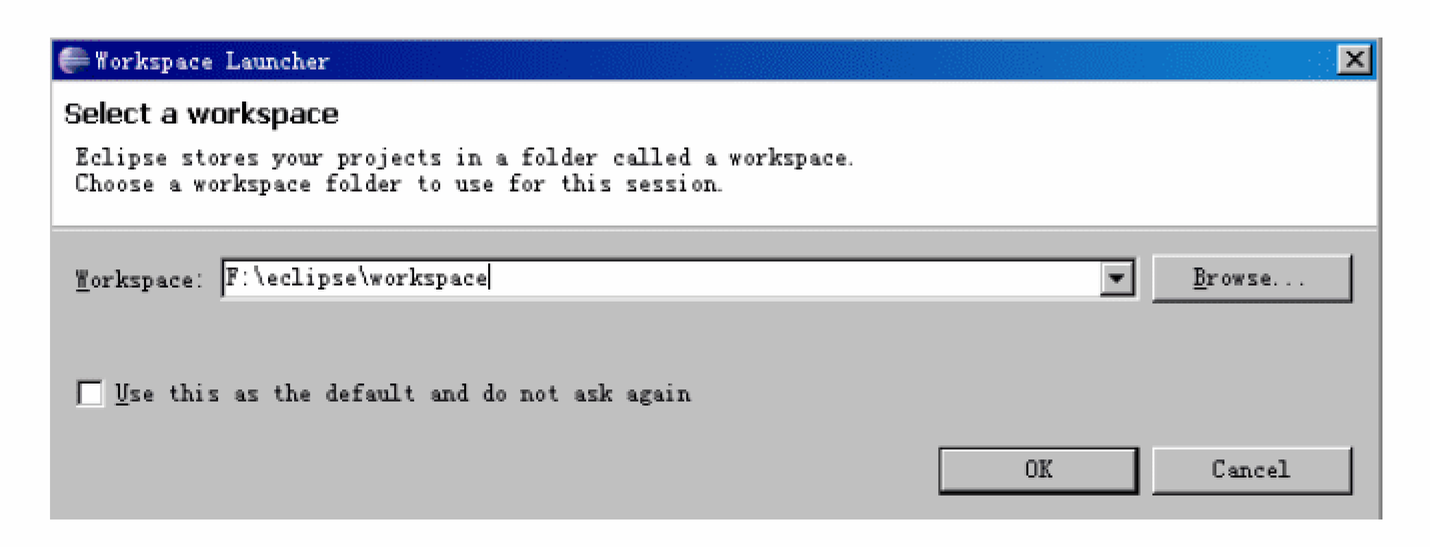


图 1-18　选择工作空间

此时单击 OK 按钮，完成 Eclipse 的安装。

3．安装Android SDK

完成 JDK 和 Eclipse 的安装后，接下来需要下载安装 Android SDK，具体步骤如下。

(1) 打开 Android 开发者社区网址 http://developer.android.com/，然后转到 SDK 下载页面(网址是 http://developer.android.com/sdk/index.html)，如图 1-19 所示。



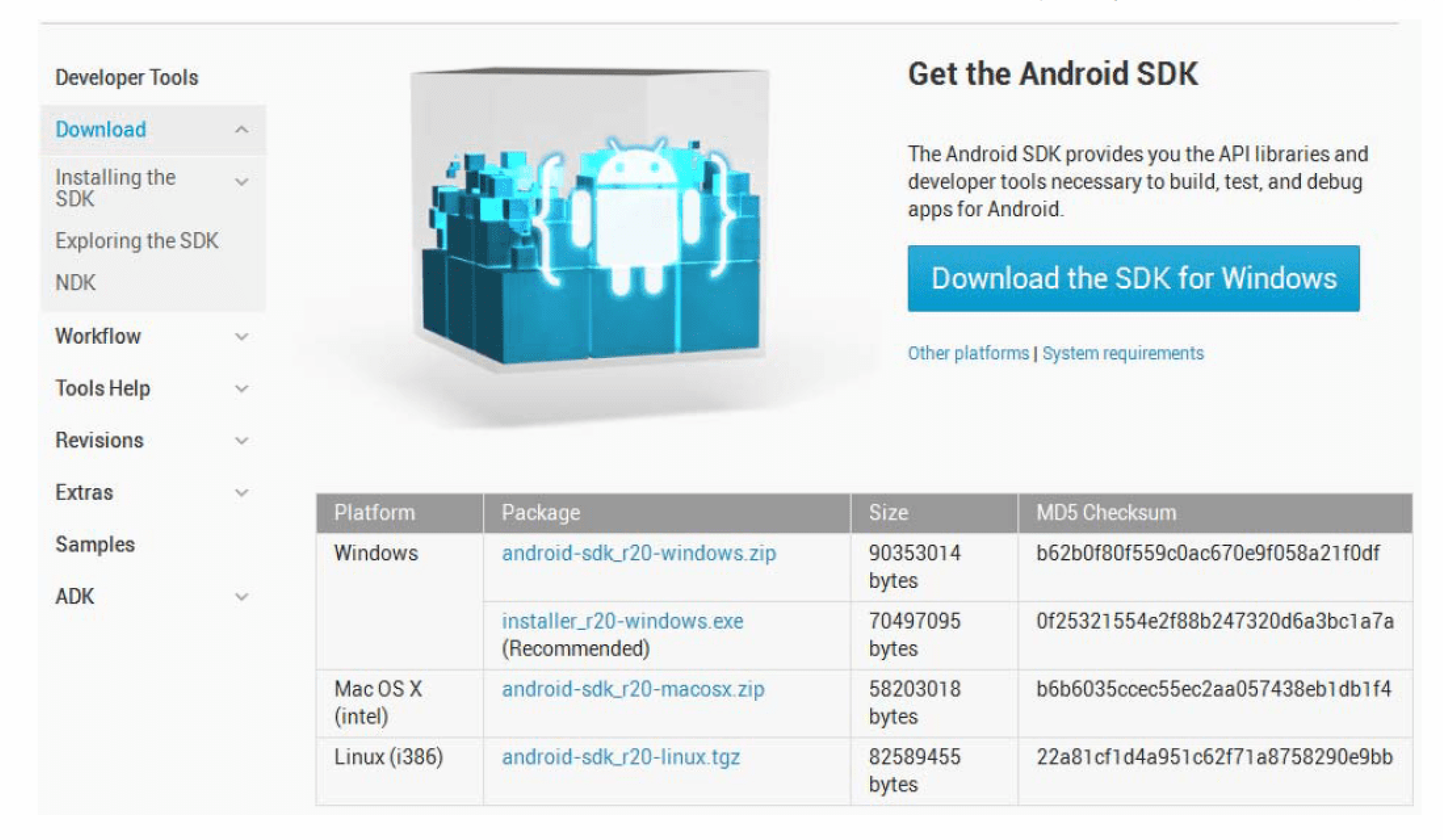


图 1-19　SDK下载页面

(2) 在此选择用于 Windows 平台的链接 android-sdk_r20-windows.zip，弹出如图 1-20 所示的对话框。

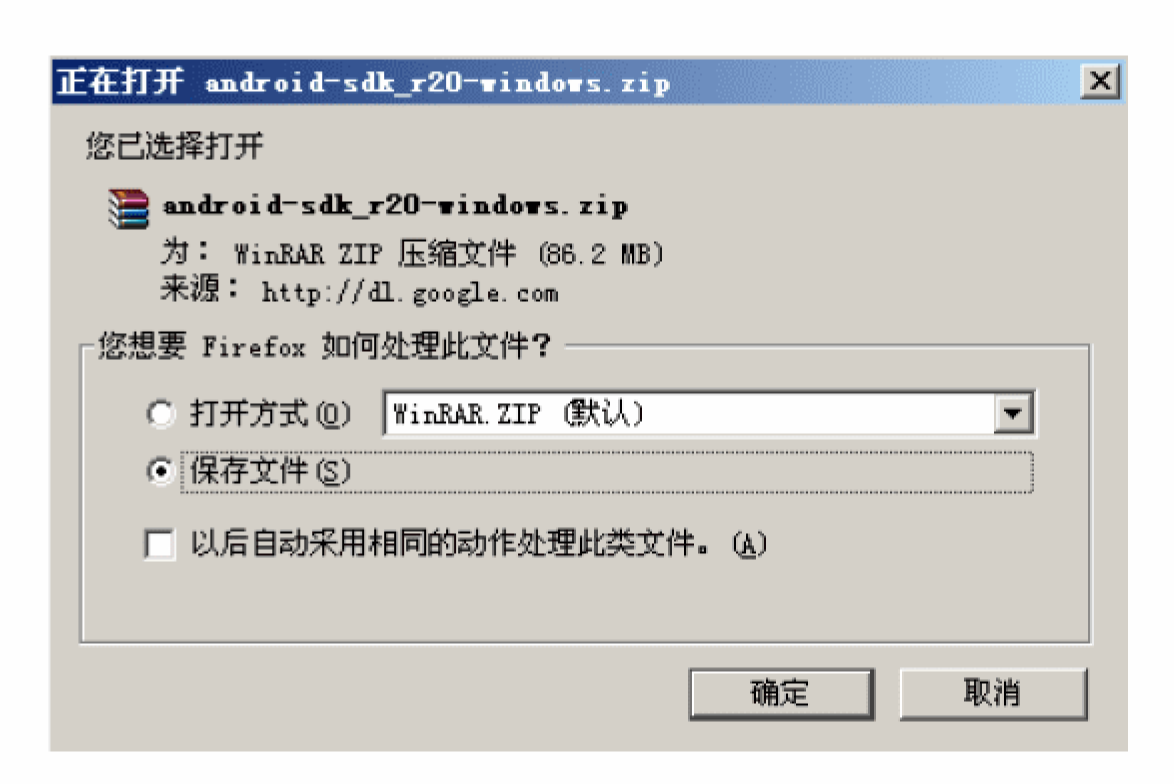


图 1-20　Android SDK下载页面

(3) 下载后解压压缩文件，假设下载后的文件解压存放在 F:\android\目录下，并将其 tools 目录的绝对路径添加到系统的 PATH 中，具体操作步骤如下。

① 右击“我的电脑”图标，选择“属性”|“高级”命令，然后单击“环境变量”，在“系统变量”处选择新建，在“变量名”处输入“SDK_HOME”，在“变量值”中输入刚才的目录，比如笔者的目录是 F:\android-sdk-windows，如图 1-21 所示。

② 找到 PATH 的变量，双击或单击编辑，在“变量值”最前面加上“%SDK_HOME%\tools;”，如图 1-22 所示。

③ 依次选择“开始”|“运行”命令，在“运行”对话框中输入“cmd”并按 Enter 键，在打开的 CMD 窗口中输入一个测试命令，例如，输入“android –h”，如果显示如图 1-23 所示的提示信息，则说明安装成功。



图 1-21　设置系统变量

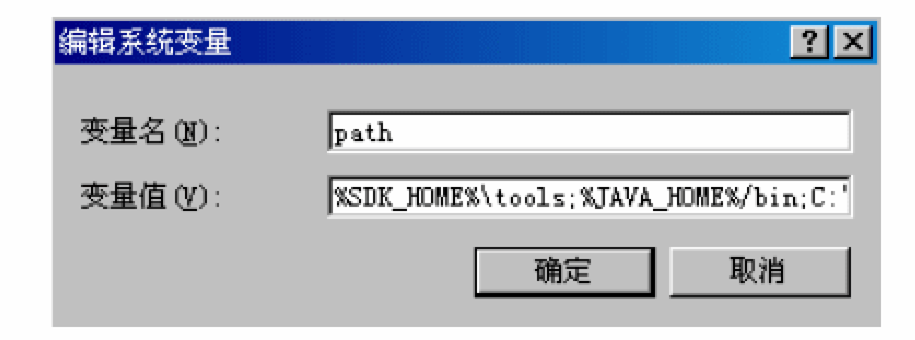


图 1-22　设置系统变量

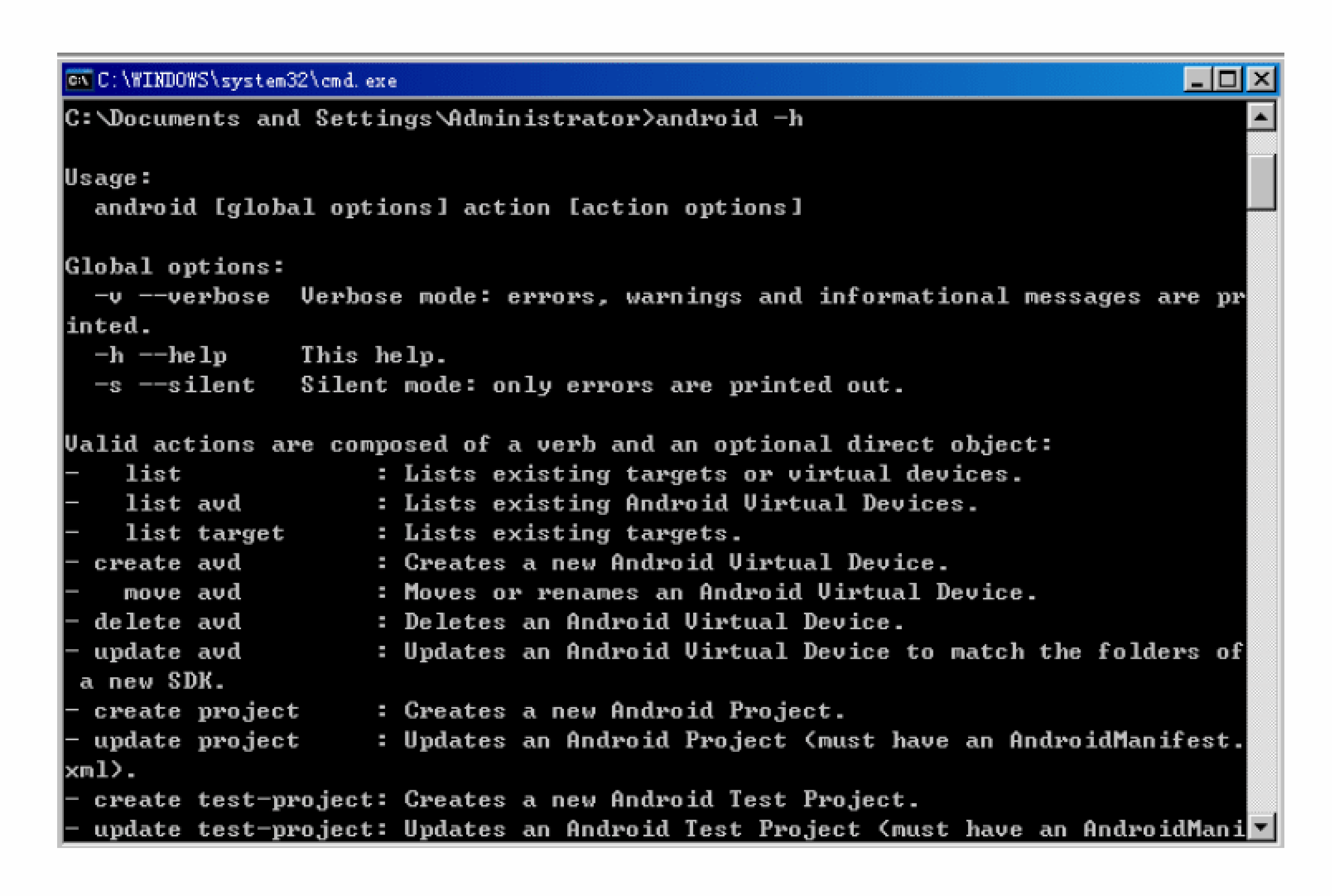


图 1-23　设置系统变量

4．安装ADT

Android 为 Eclipse 定制了一个专用插件 Android Development Tools(ADT)，此插件为用户提供了一个开发 Android 应用程序的综合环境。ADT 扩展了 Eclipse 的功能，可以让用户快速地建立 Android 项目，创建应用程序界面。要安装 Android Development Tools plug-in，需要首先打开 Eclipse IDE，然后进行如下操作。

(1) 打开 Eclipse 后，依次选择菜单栏中的 Help | Install New Software 命令，如图 1-24 所示。

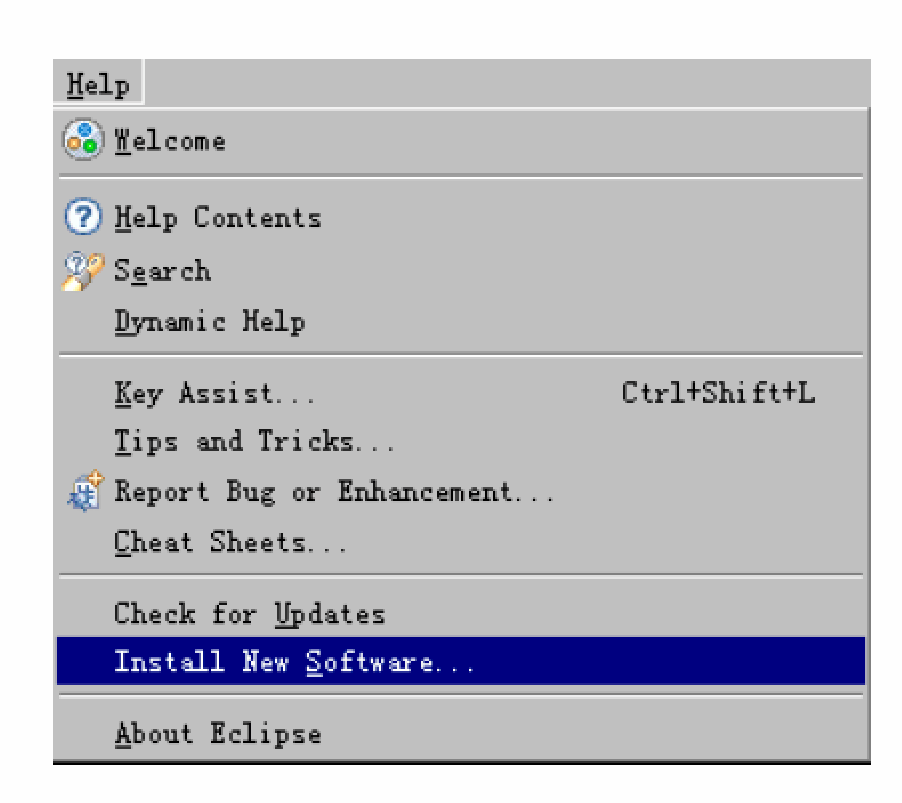


图 1-24　选择Install New Software命令

(2) 在弹出的 Install 对话框中单击 Add 按钮，如图 1-25 所示。



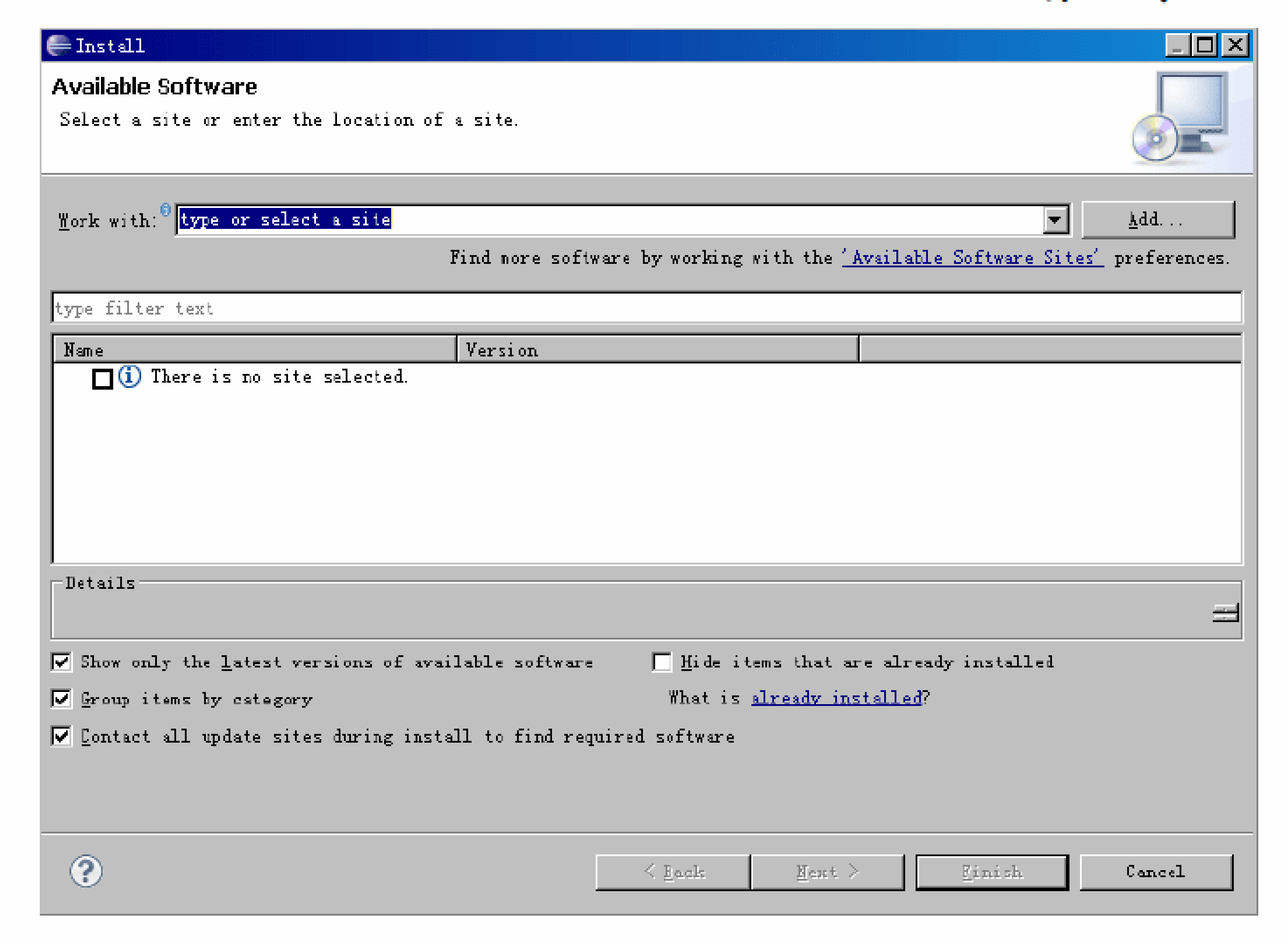


图 1-25　添加插件

(3) 在弹出的 Add Site 对话框中分别输入名字和地址。名字可以自己命名，例如“123”，但是在 Location 文本框中必须输入插件的网络地址 http://dl-ssl.google.com/Android/eclipse/，然后单击 OK 按钮，如图 1-26 所示。

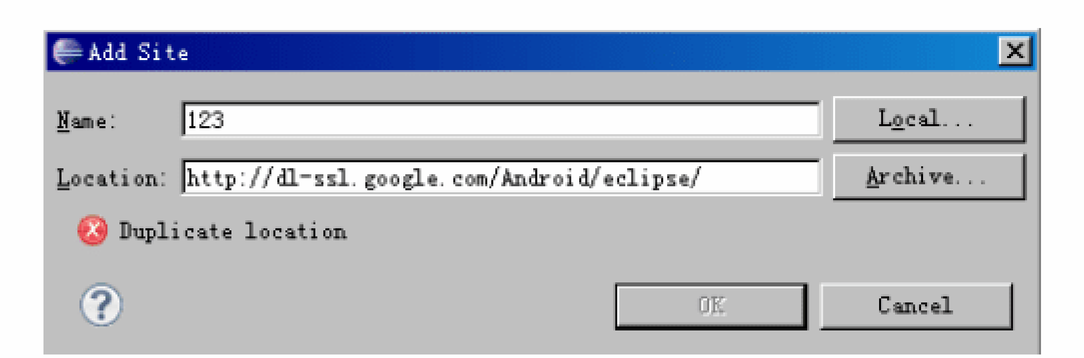


图 1-26　设置地址

(4) 此时在 Install 对话框中将会显示系统中可用的插件，如图 1-27 所示。

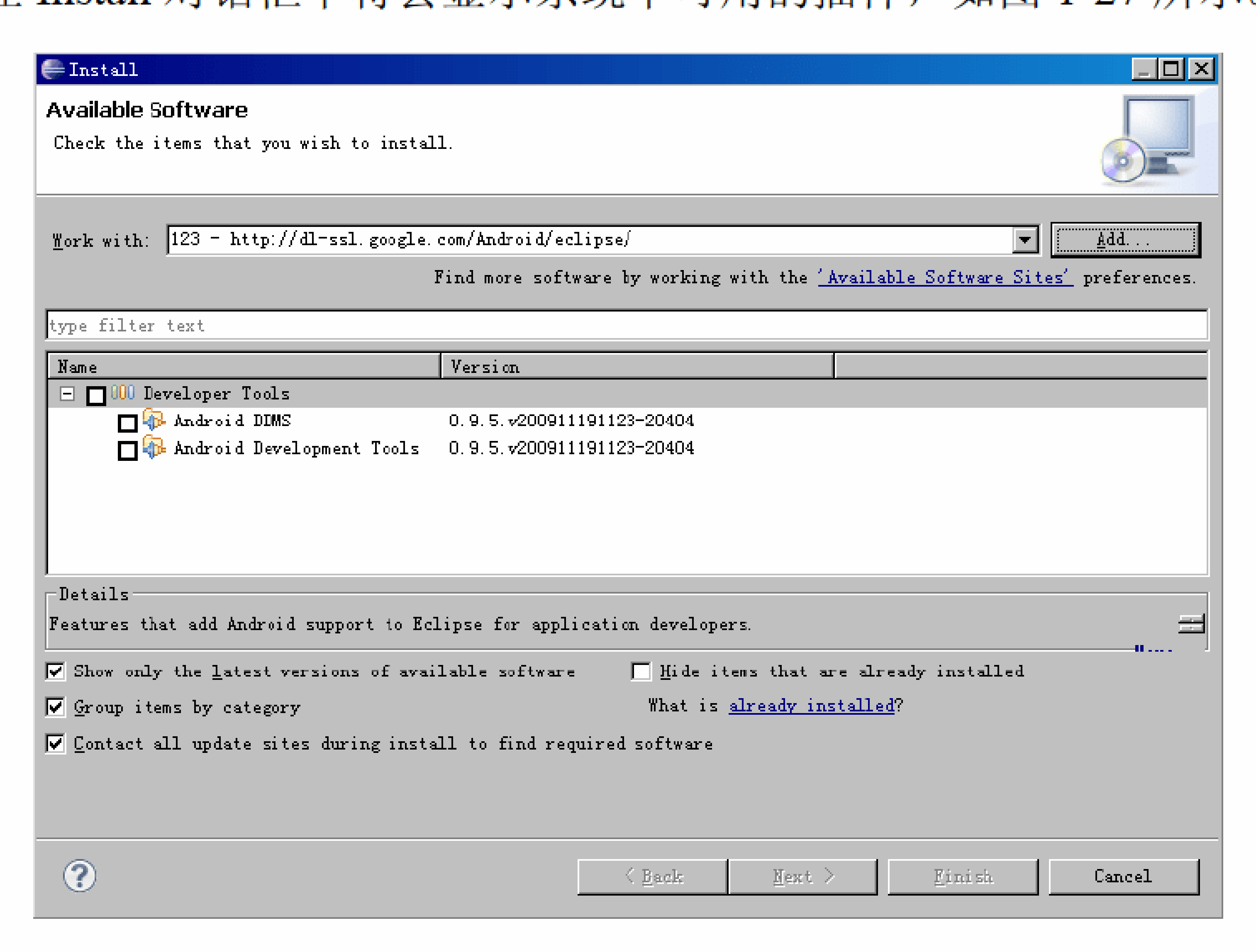


图 1-27　插件列表

(5) 选中 Android DDMS 和 Android Development Tools 插件，然后单击 Next 按钮，进入如图 1-28 所示的对话框。

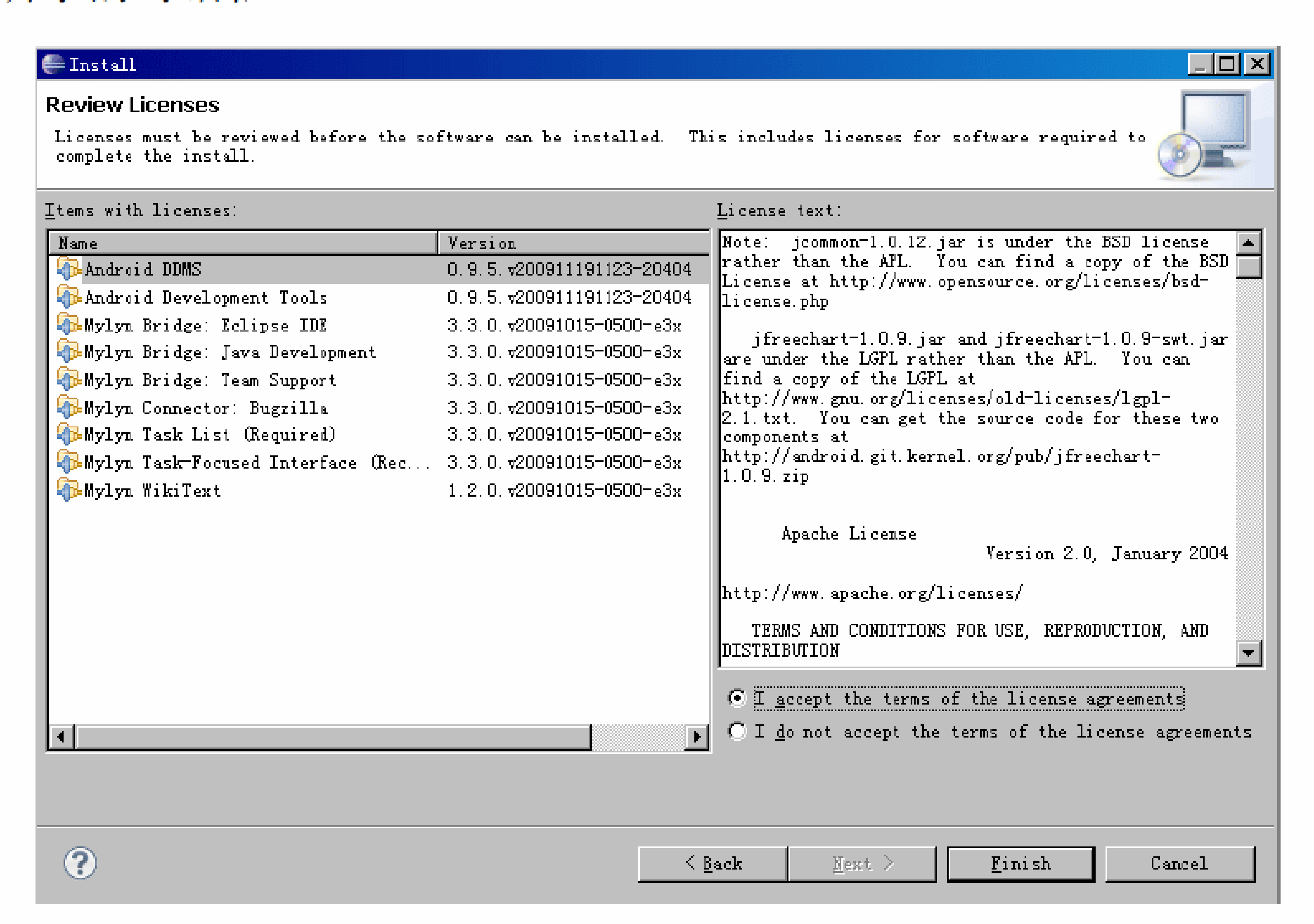


图 1-28　Review Licenses界面

(6) 选中 I accept the terms of the license agreements 单选按钮，然后单击 Finish 按钮，开始进行安装，如图 1-29 所示。

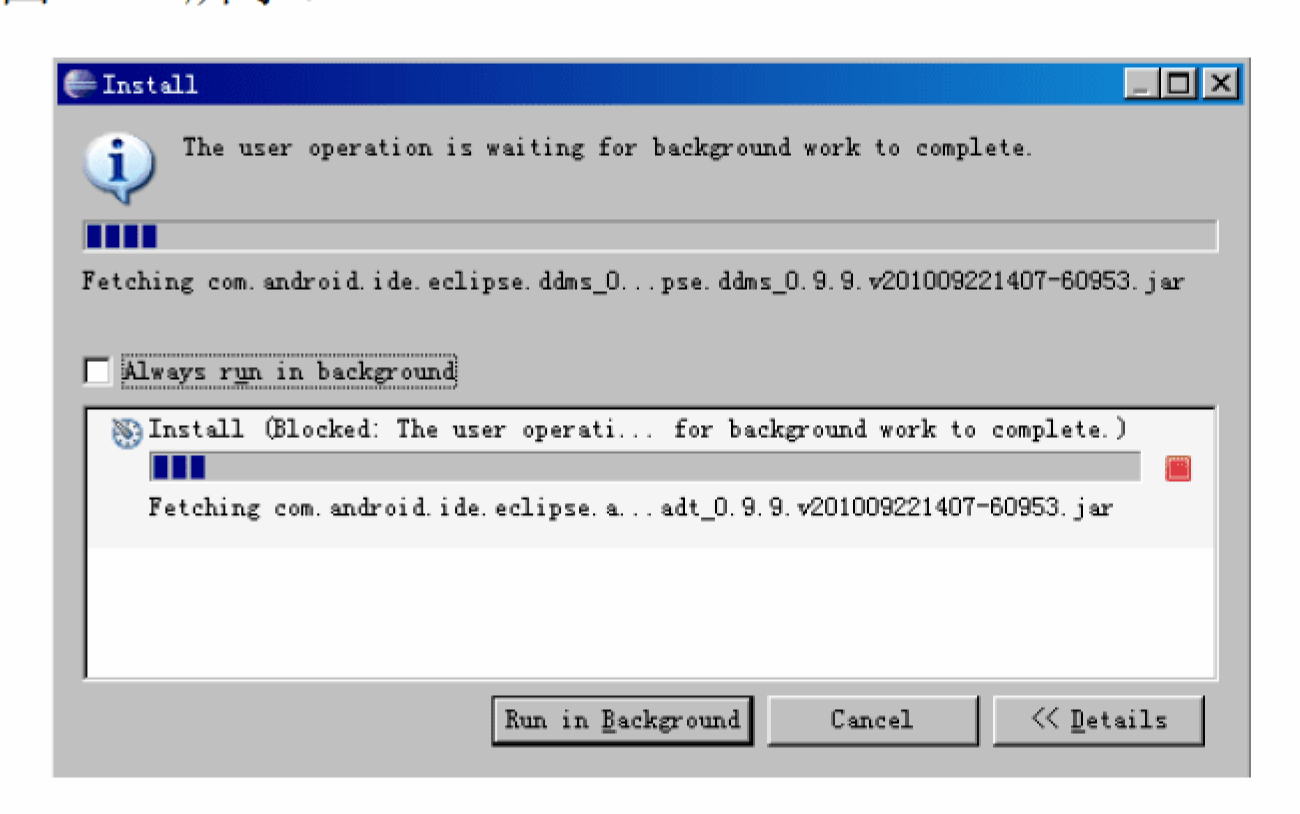


图 1-29　开始安装

注意：在上述步骤中，可能会发生“计算插件占用资源”的情况，整个计算过程有点慢。完成后会提示重启 Eclipse 来加载插件，重启之后就可以使用了。并且不同版本的 Eclipse 安装插件的方法和步骤是不同的，但是都大同小异，读者可以根据操作提示自行解决。

1.3.3　设定Android SDK主目录

当完成上述插件的安装工作后，还不能使用 Eclipse 创建 Android 项目，还需要在 Eclipse 中设置 Android SDK 的主目录。

(1) 打开 Eclipse，在菜单栏中依次选择 Window | Preferences 命令，如图 1-30 所示。



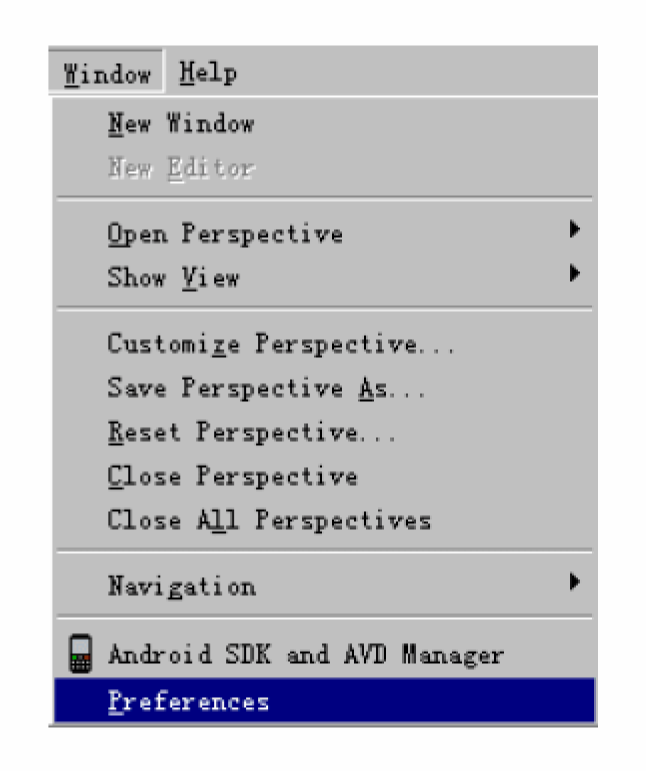


图 1-30　选择Preferences命令

(2) 在弹出的对话框的左侧选中 Android 选项，在右侧设定 Android SDK 所在目录为 F:\android-sdk-windows，然后单击 OK 按钮完成设置，如图 1-31 所示。

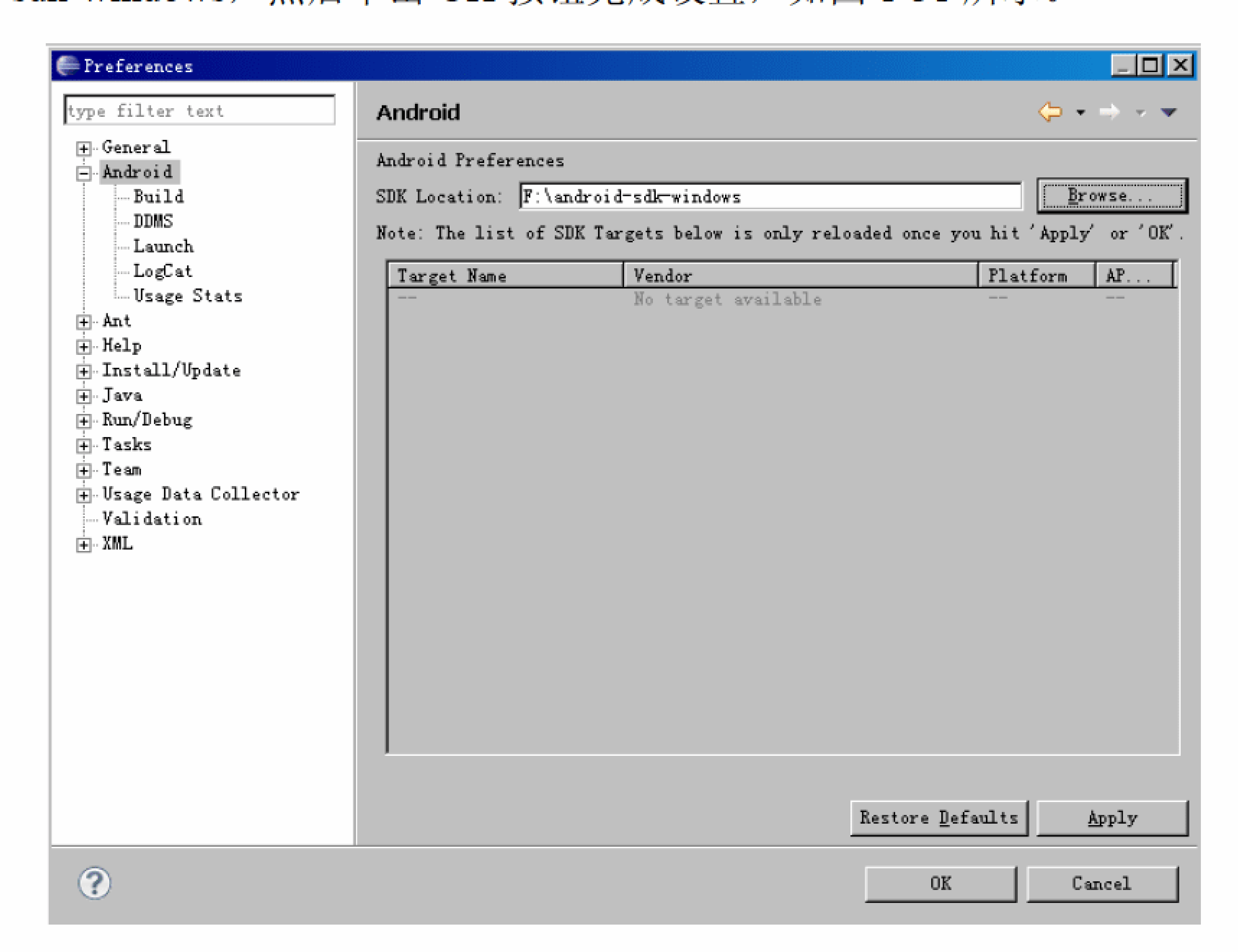


图 1-31　Preferences对话框

1.4　创建 Android 虚拟设备

我们都知道程序开发需要调试，只有经过调试之后才能知道程序是否正确运行。作为一款手机系统，我们如何在电脑平台之上调试 Android 程序呢？不用担心，Google 为我们提供了模拟器来解决这个问题。所谓模拟器，就是指在电脑上模拟 Android 系统，然后用这个模拟器来调试并运行开发的 Android 程序。开发人员无须用一个真实的 Android 手机，只需通过电脑模拟运行一个手机，即可开发出应用在手机上面的程序。Android 模拟器在电脑上的运行效果如图 1-32 所示。

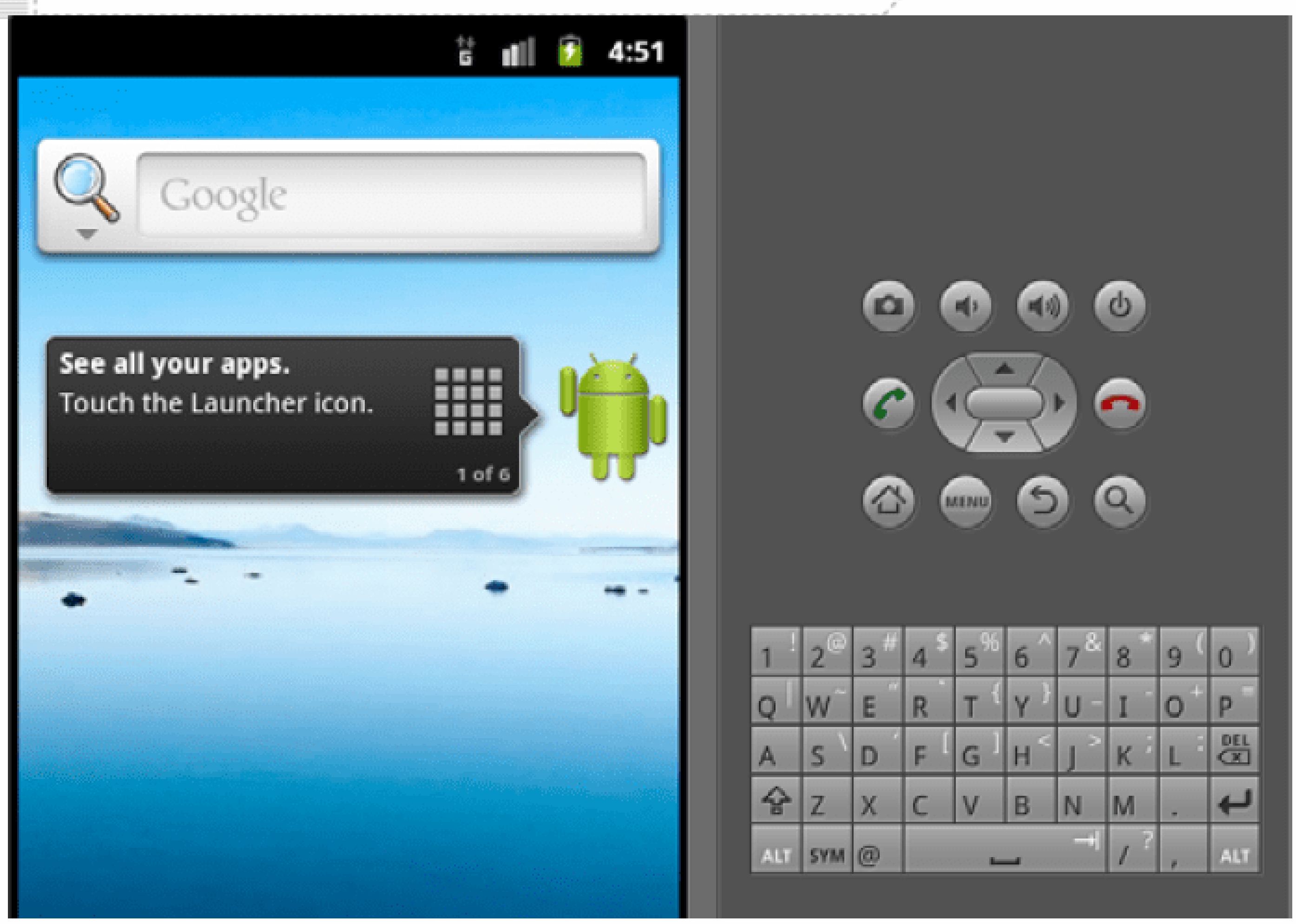


图 1-32　模拟器

1.4.1　Android模拟器简介

Android 模拟器的全称是 Android Virtual Device，简称为 AVD。对于 Android 程序的开发者来说，模拟器的推出给开发者带来了极大的方便，无论是在开发工作上还是在测试工作上，无论在 Windows 环境下还是在 Linux 环境下，Android 模拟器都可以顺利运行。并且官方提供了 Eclipse 插件，可以将模拟器集成到 Eclipse 的 IDE 环境。当然，也可以从命令行启动 Android 模拟器。

获取模拟器的方法非常简单，既可以从官方站点(http://developer.Android.com/)免费下载单独的模拟器，也可以先下载 Android SDK，解压后在其 SDK 的根目录下有一个名为 tools 的文件夹，此文件夹下包含了完整的模拟器和一些非常有用的工具。

Android SDK 中包含的模拟器的功能非常齐全，电话本、通话等功能都可正常使用(当然你没办法真的从这里打电话)。甚至其内置的浏览器和 Maps 都可以联网。用户可以通过键盘输入，或者通过鼠标点击模拟器按键输入，甚至还可以使用鼠标点击、拖动屏幕进行操纵。

1.4.2　模拟器和真机的区别

Android 模拟器不能完全替代真机，具体来说它们有如下差异。

- 模拟器不支持呼叫和接听实际来电，但可以通过控制台模拟电话呼叫(呼入和呼出)。
- 模拟器不支持 USB 连接。
- 模拟器不支持相机/视频捕捉。
- 模拟器不支持音频输入(捕捉)，但支持输出(重放)。
- 模拟器不支持扩展耳机。
- 模拟器不能确定连接状态。
- 模拟器不能确定电池电量状态和交流充电状态。



- ❑ 模拟器不能确定 SD 卡的插入/弹出。
- ❑ 模拟器不支持蓝牙。

1.4.3　创建Android虚拟设备

每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核、系统图像和数据分区，还可以有自己的 SD 卡、用户数据以及外观显示等。创建 AVD 的基本步骤如下。

(1) 单击 Eclipse 菜单中的图标，如图 1-33 所示。

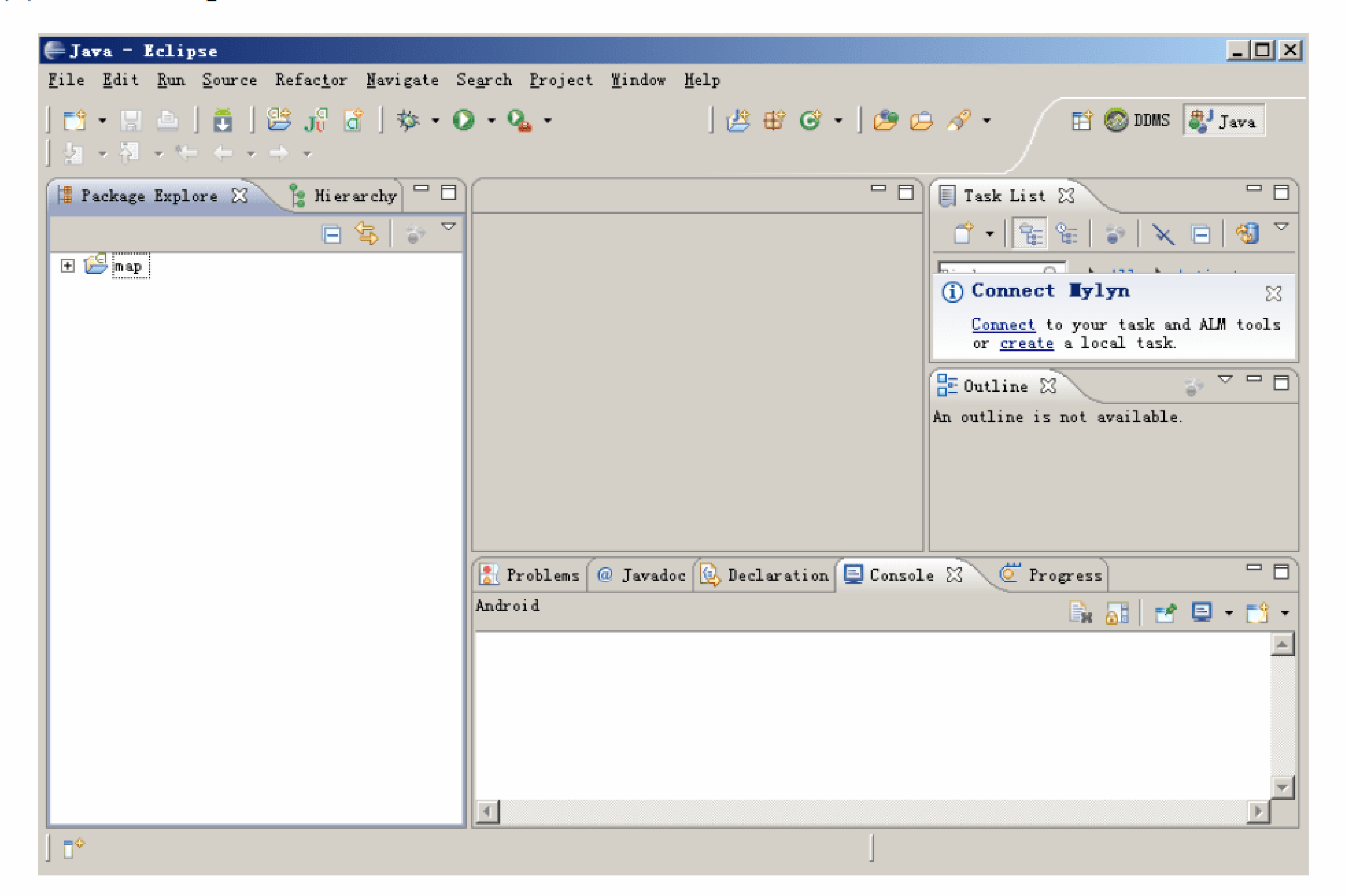


图 1-33　Eclipse界面

(2) 弹出 Android SDK and AVD Manager 界面，在该界面的左侧选择 Virtual devices 选项，如图 1-34 所示。

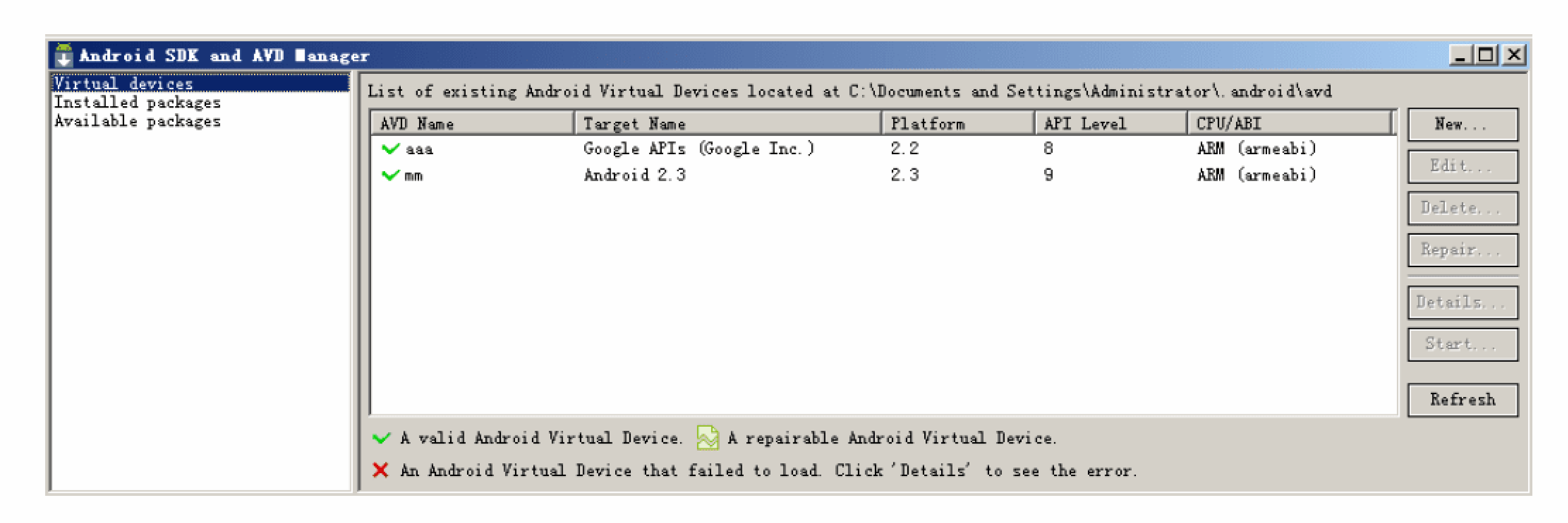


图 1-34　Android SDK and AVD Manager界面

在 Virtual devices 设置界面中列出了当前已经安装的 AVD 版本，我们可以通过界面右

侧的按钮来创建、删除或修改 AVD。主要按钮的具体说明如下。

- New...：创建新的 AVD，单击此按钮在弹出的界面中可以创建一个新 AVD，如图 1-35 所示。
- Edit...：修改已经存在的 AVD。
- Delete...：删除已经存在的 AVD。
- Start...：启动一个 AVD 模拟器。

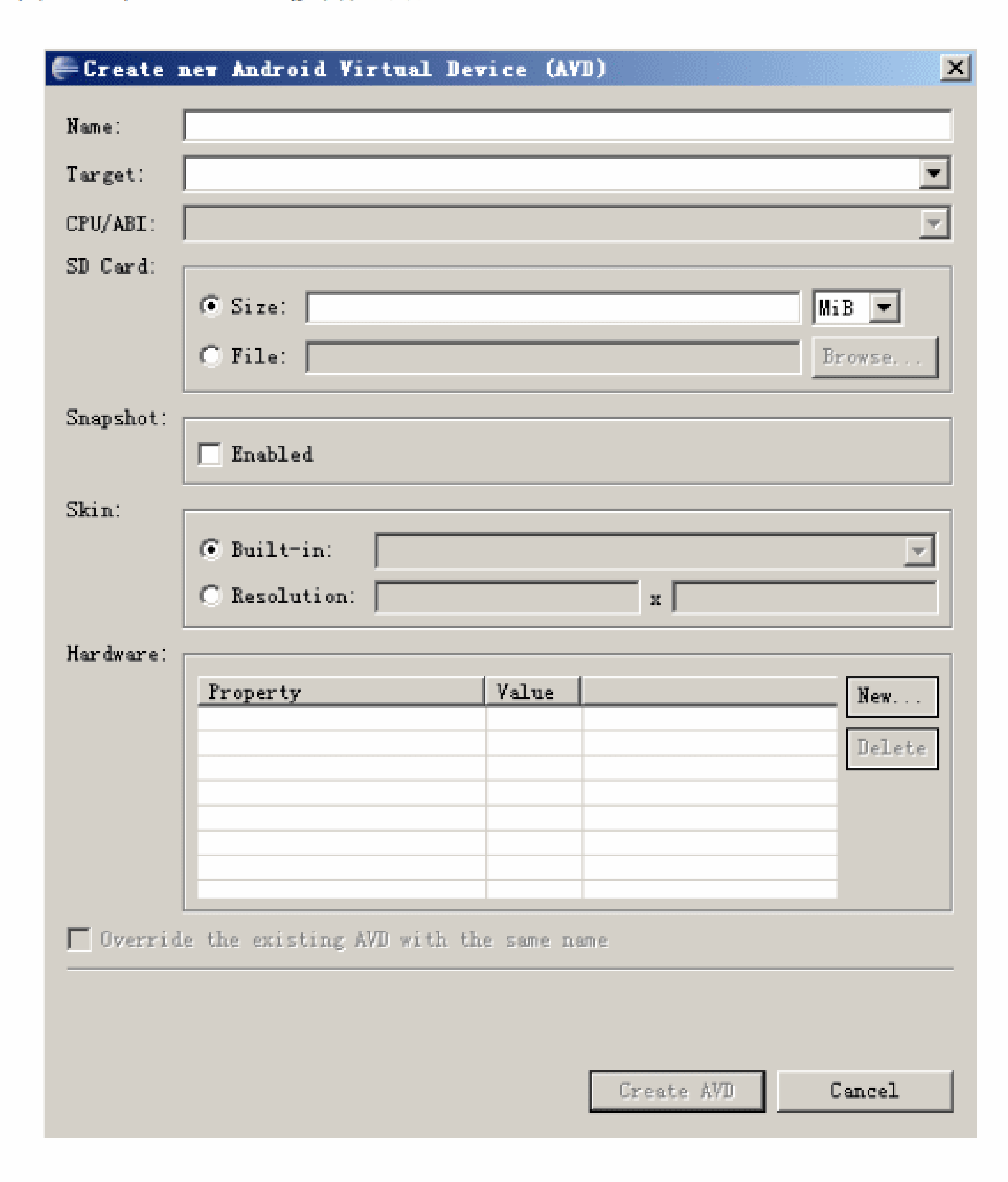


图 1-35　新建AVD界面

注意：我们也可以在 CMD 中创建或删除 AVD，例如可以按照如下 CMD 命令创建一个 AVD。

```
android create avd --name <your_avd_name> --target <targetID>
```

其中“your_avd_name”是需要创建的 AVD 的名字，在 CMD 窗口中的显示如图 1-36 所示。

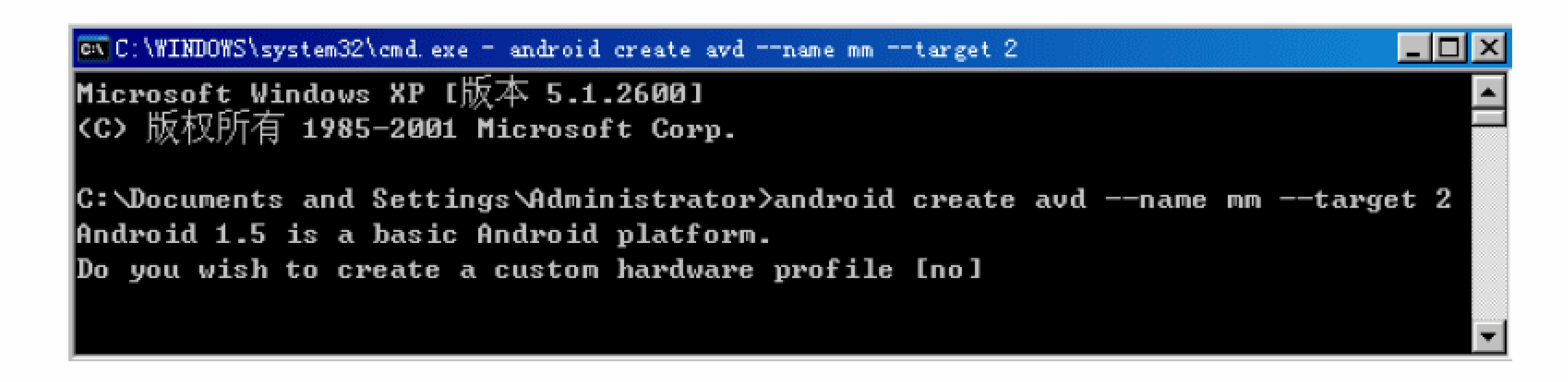


图 1-36　CMD窗口

1.4.4　启动AVD模拟器

在调试的时候需要启动 AVD 模拟器。启动 AVD 模拟器的基本流程如下。



(1) 选择图 1-34 所示列表中名为“mm”的 AVD，单击 Start 按钮，弹出 Launch Options 对话框，如图 1-37 所示。

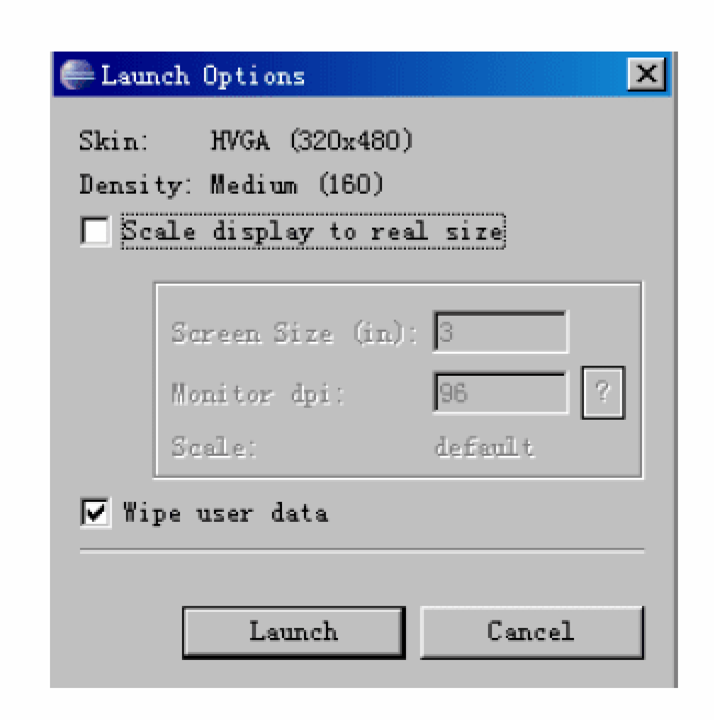


图 1-37　Launch Options对话框

(2) 单击 Launch 按钮将会运行名为“mm”的模拟器，如图 1-38 所示。

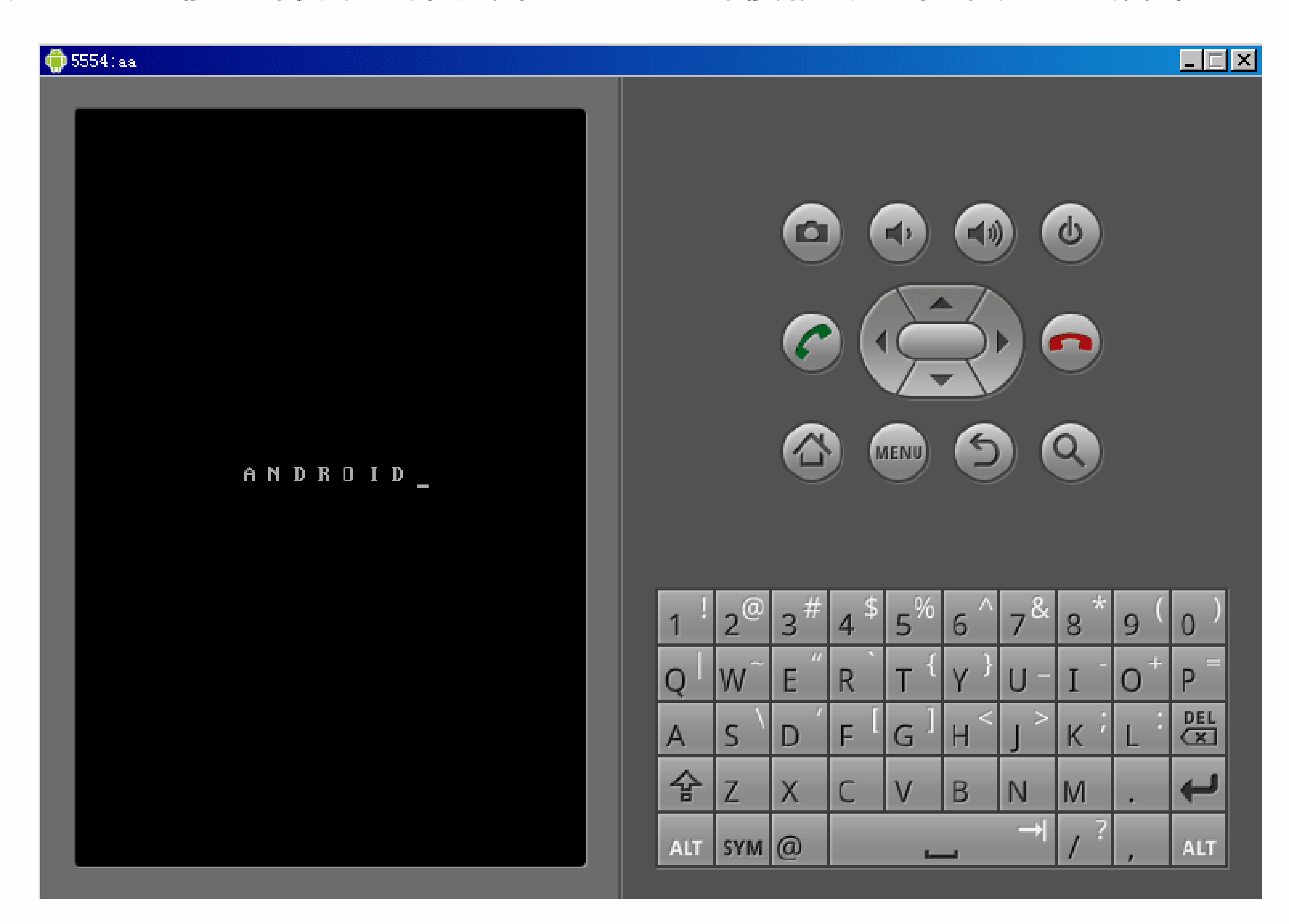


图 1-38　模拟器运行成功

1.4.5　快速安装SDK的方法

通过 Android SDK Manager 在线安装的速度非常慢，而且有时容易挂掉。其实可以先从网络中寻找 SDK 资源，用迅雷等下载工具下载后，将其放到指定目录即可完成安装。具体方法是先下载 android-sdk-windows(该软件需能更新)，然后在 android-sdk-windows 下双击 setup.exe，在更新的过程中会发现安装 Android SDK 的速度是 1Kb/s，此时打开迅雷，分别输入下面的地址：

```
https://dl-ssl.google.com/android/repository/platform-tools r05-
windows.zip
https://dl-ssl.google.com/android/repository/docs-3.1 r01-linux.zip
https://dl-ssl.google.com/android/repository/android-2.2_r02-windows.zip
```

```
https://dl-ssl.google.com/android/repository/android-2.3.3 r01-linux.zip
https://dl-ssl.google.com/android/repository/android-2.1 r02-windows.zip
https://dl-ssl.google.com/android/repository/samples-2.3.3 r01-linux.zip
https://dl-ssl.google.com/android/repository/samples-2.2 r01-linux.zip
https://dl-ssl.google.com/android/repository/samples-2.1 r01-linux.zip
https://dl-ssl.google.com/android/repository/compatibility r02.zip
https://dl-ssl.google.com/android/repository/tools r11-windows.zip
https://dl-ssl.google.com/android/repository/google apis-10 r02.zip
https://dl-ssl.google.com/android/repository/android-2.3.1 r02-linux.zip
https://dl-ssl.google.com/android/repository/usb_driver_r04-windows.zip
https://dl-ssl.google.com/android/repository/googleadmobadssdkandroid-4.1.0.zip
https://dl-ssl.google.com/android/repository/market_licensing-r01.zip
https://dl-ssl.google.com/android/repository/market billing r01.zip
https://dl-ssl.google.com/android/repository/google apis-8 r02.zip
https://dl-ssl.google.com/android/repository/google apis-7 r01.zip
https://dl-ssl.google.com/android/repository/google apis-9 r02.zip
…
可以继续根据自己的开发要求选择不同版本的 API
```

下载完后将它们复制到 android-sdk-windows/Temp 目录下，然后再运行 setup.exe。选中需要的 API 选项，即可安装。记得把原始文件保留好，因为放在 Temp 目录下的文件安装好后就没有了。

1.5 搭建环境过程中的常见问题

在搭建完成开发环境后，接下来将总结在搭建 Android SDK 环境过程中出现过的问题，帮助读者在实践中快速成功搭建 Android 开发环境。

1. 不能在线更新

在安装 Android 后，需要更新为最新的资源和配置，但是在启动 Android 后，经常会不能更新，弹出如图 1-39 所示的错误提示。

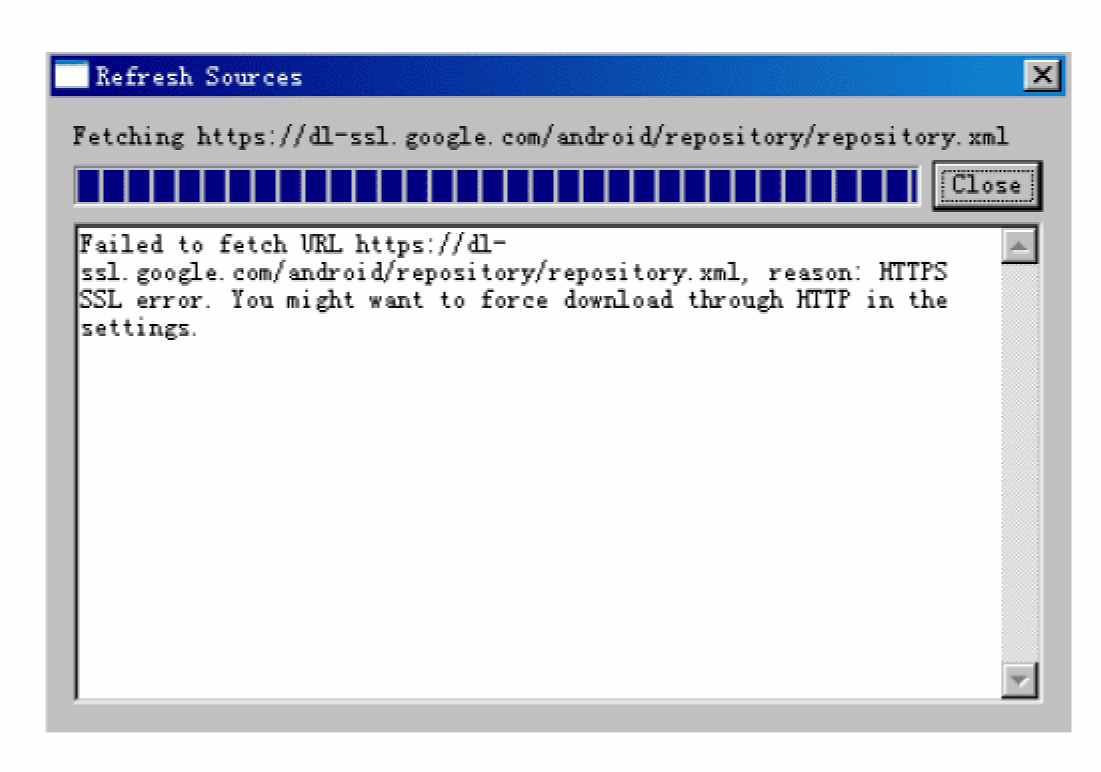


图 1-39　不能更新的错误提示

Android 默认的在线更新地址是 https://dl-ssl.google.com/android/eclipse/，但是经常会出现错误。如果此地址不能更新，可以自行设置更新地址，修改为 http://dl-



ssl.google.com/android/repository/repository.xml。具体操作方法如下。

(1) 单击 Android SDK and AVD Manager 对话框左侧的 Available Packages 选项，然后单击 Add Site…按钮，如图 1-40 所示。

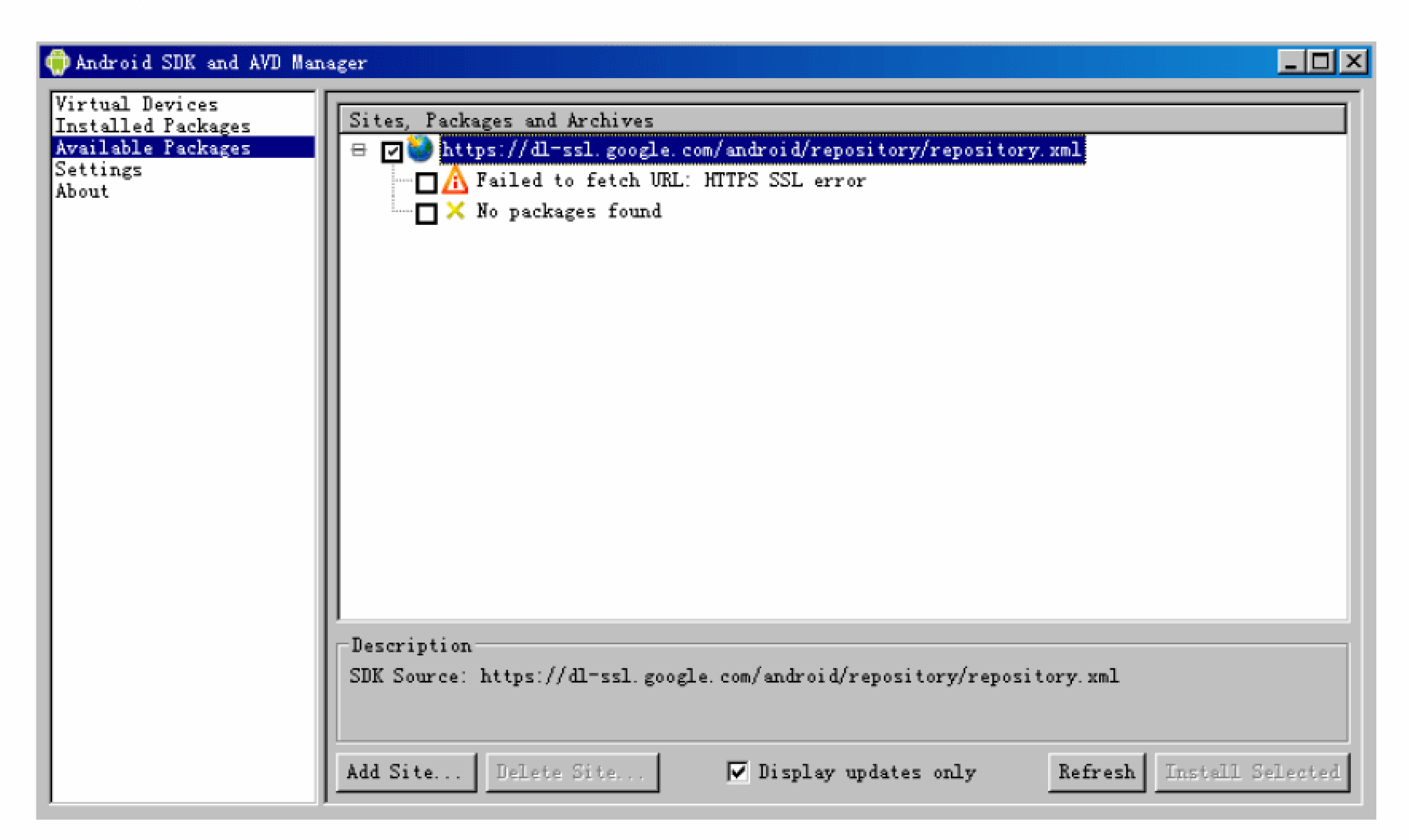


图 1-40　Android SDK and AVD Manager对话框

(2) 在弹出的 Add Site URL 对话框中输入如下修改后的地址，如图 1-41 所示。

```
http://dl-ssl.google.com/android/repository/repository.xml
```

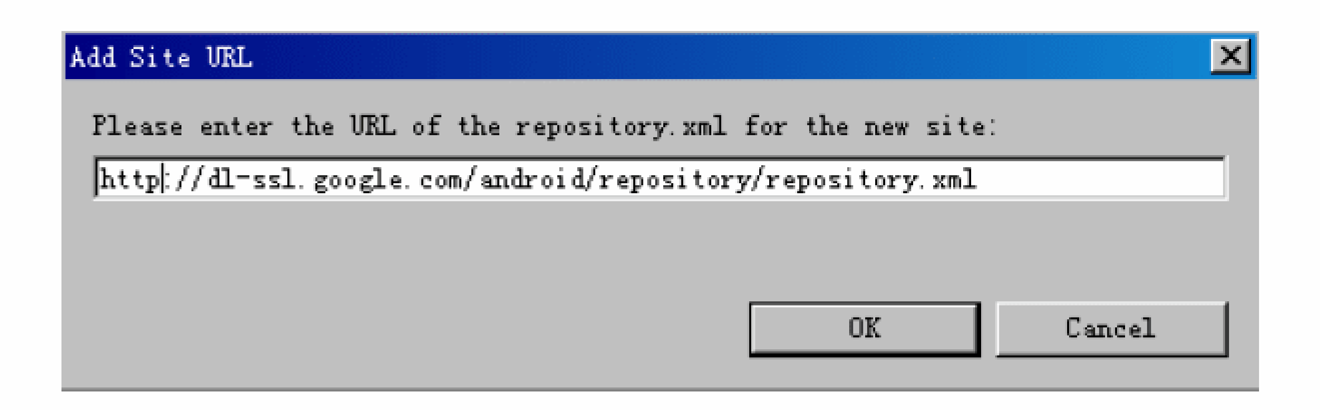


图 1-41　Add Site URL对话框

(3) 单击 OK 按钮完成设置，此时就可以使用更新功能了，如图 1-42 所示。

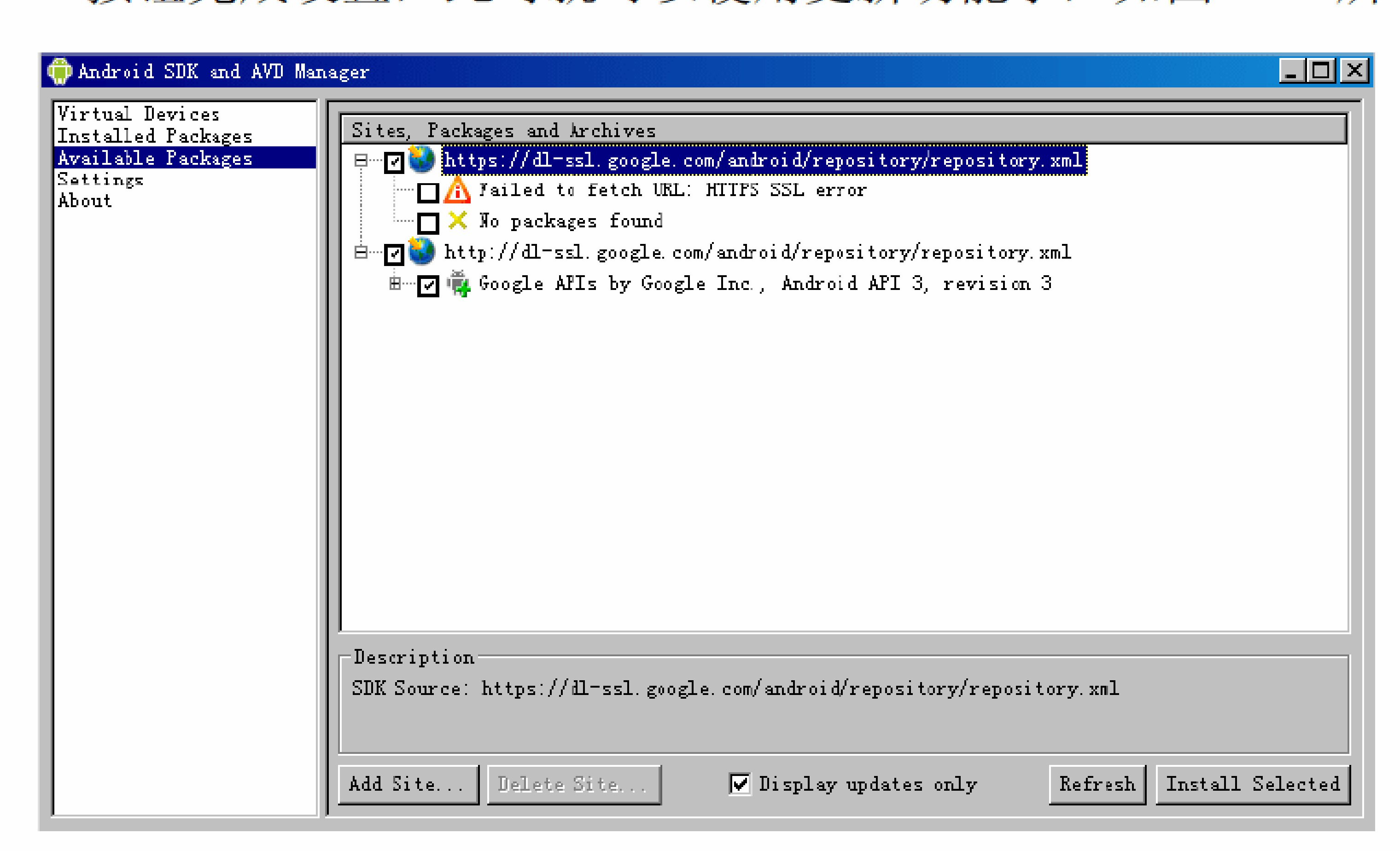


图 1-42　可以使用更新功能了

2. 显示“Project name must be specified”提示

在 Eclipse 中新建 Android 工程时，一直显示“Project name must be specified”提示，如图 1-43 所示。

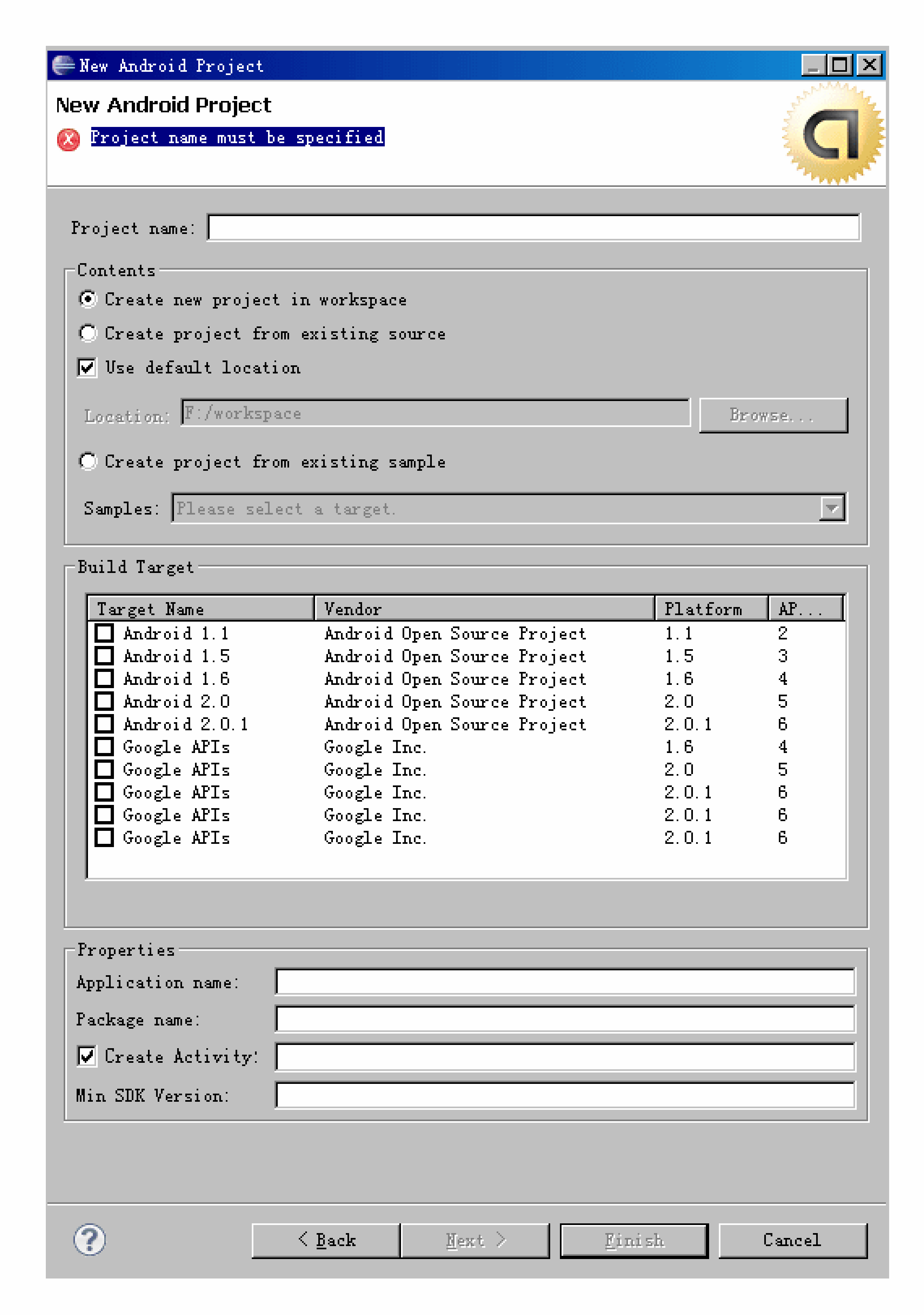


图 1-43　New Android Project对话框

造成上述问题的原因是 Android 没有更新完成，需要进行完全更新，具体方法如下。

(1) 打开 Android SDK and AVD Manager 对话框，选择左侧的 Installed Packages 选项，如图 1-44 所示。

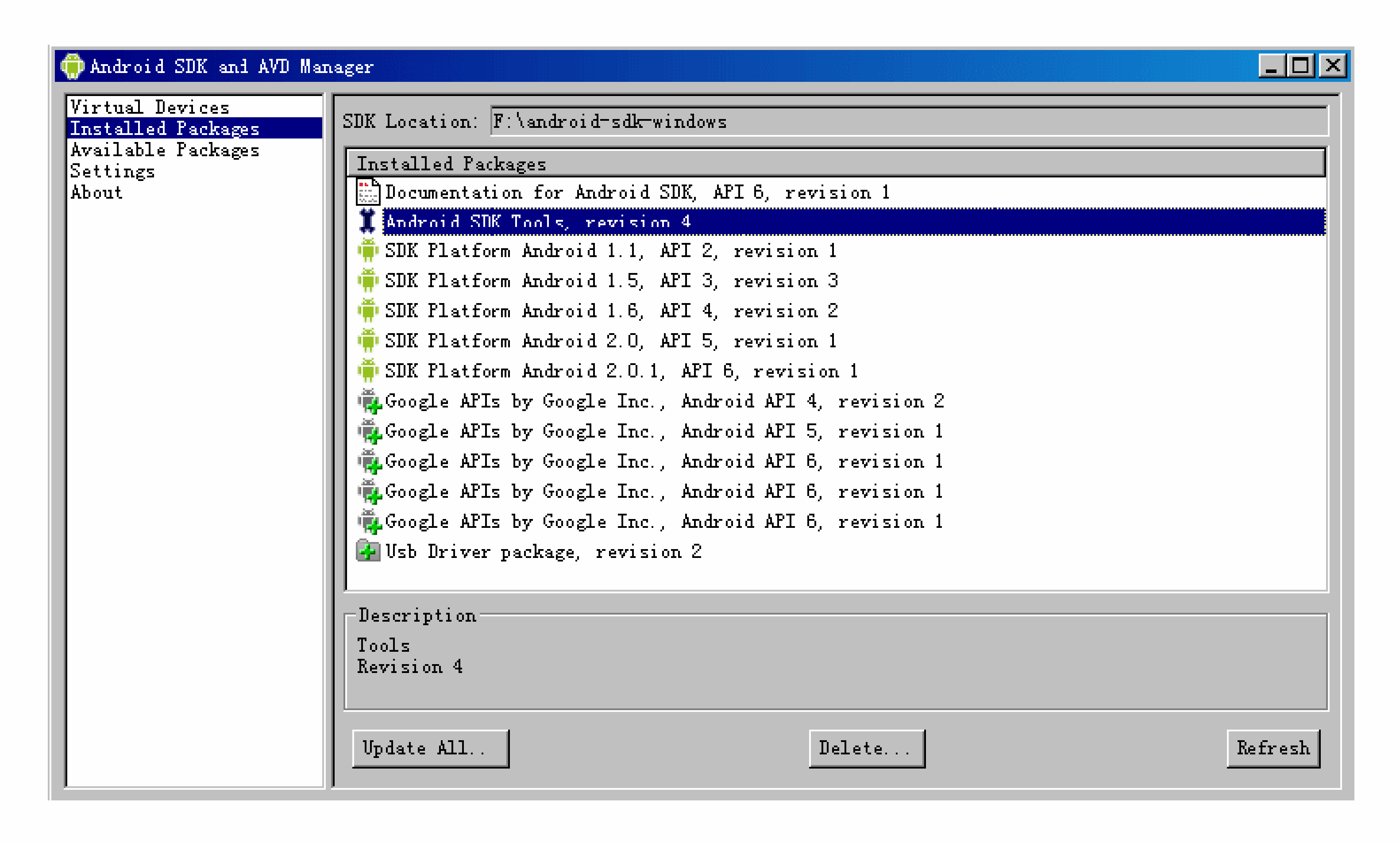


图 1-44　Android SDK and AVD Manager对话框



(2) 在如图 1-44 所示对话框的右侧列表框中选择 Android SDK Tools, revision 4 选项，在弹出的如图 1-45 所示的对话框中选中 Accept 单选按钮，然后单击 Install Accepted 按钮开始安装更新。

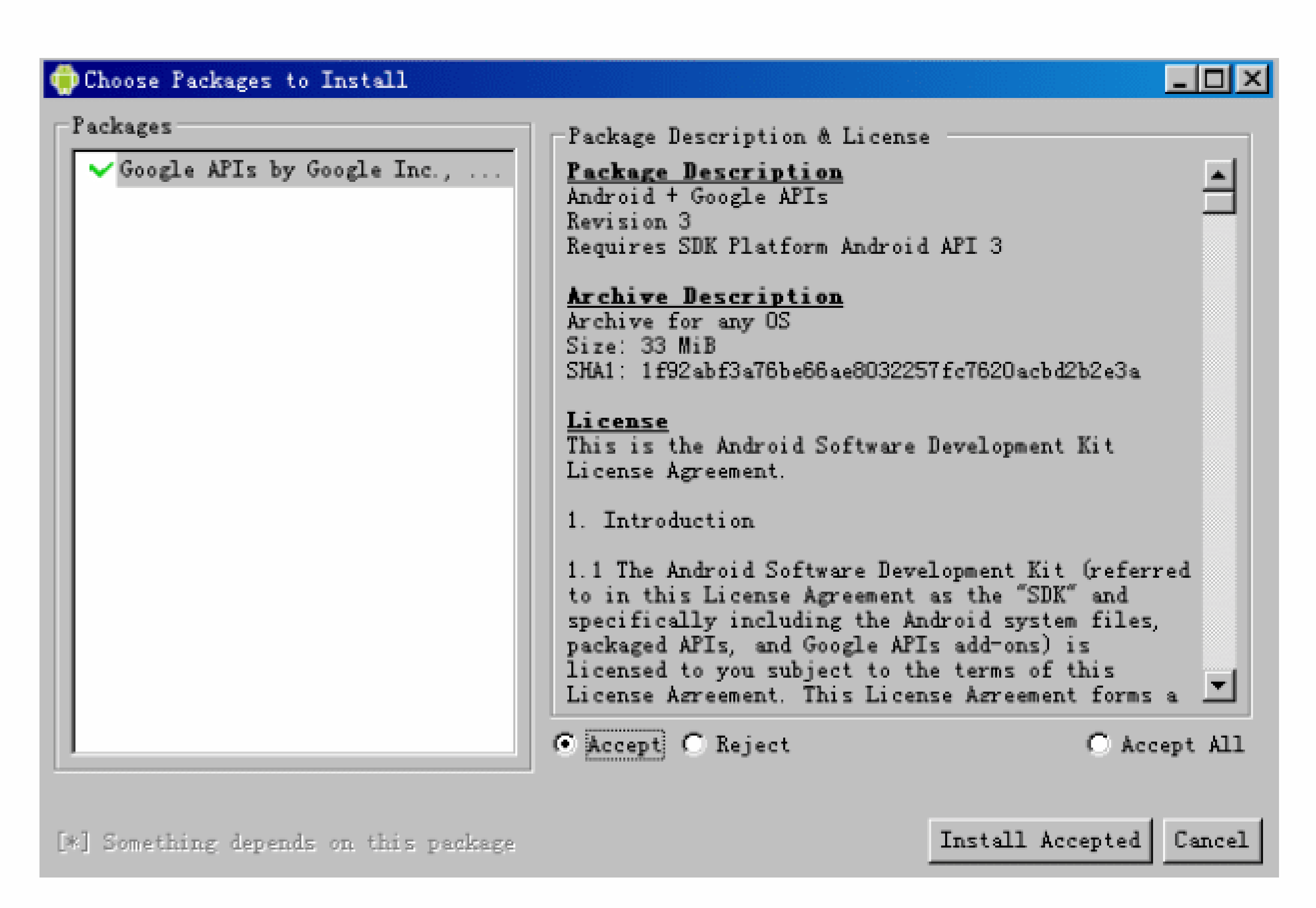


图 1-45　Choose Packages to Install对话框

3. Target列表中没有Target选项

通常来说，当 Android 开发环境搭建完毕后，在 Eclipse 工具栏中依次选择 Window | Preferences 命令，打开 Preferences 对话框，单击左侧的 Android 选项后会显示存在的 SDK Targets，如图 1-46 所示。

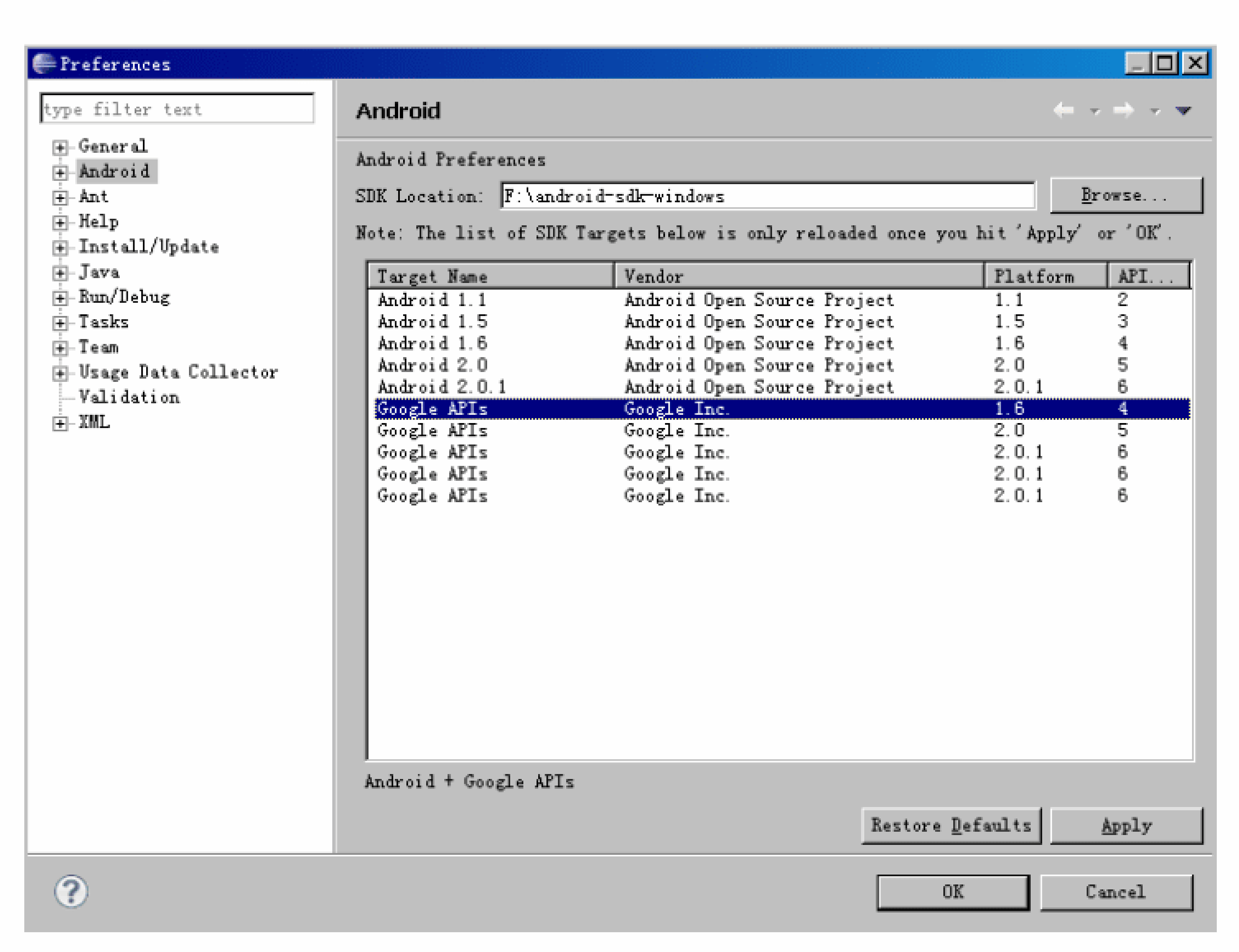


图 1-46　SDK Targets列表

可是往往会因为各种原因，而不显示 SDK Targets 列表，并且在图 1-31 所示的对话框中也不显示，并输出“Failed to find an AVD compatible with target”错误提示。

造成上述问题的原因是创建 AVD 没有成功，此时需要手工安装来解决这个问题，当然前提是 Android 已更新完毕。具体解决方法如下。

(1) 在“运行”对话框中输入“CMD”，打开 CMD 窗口，如图 1-47 所示。

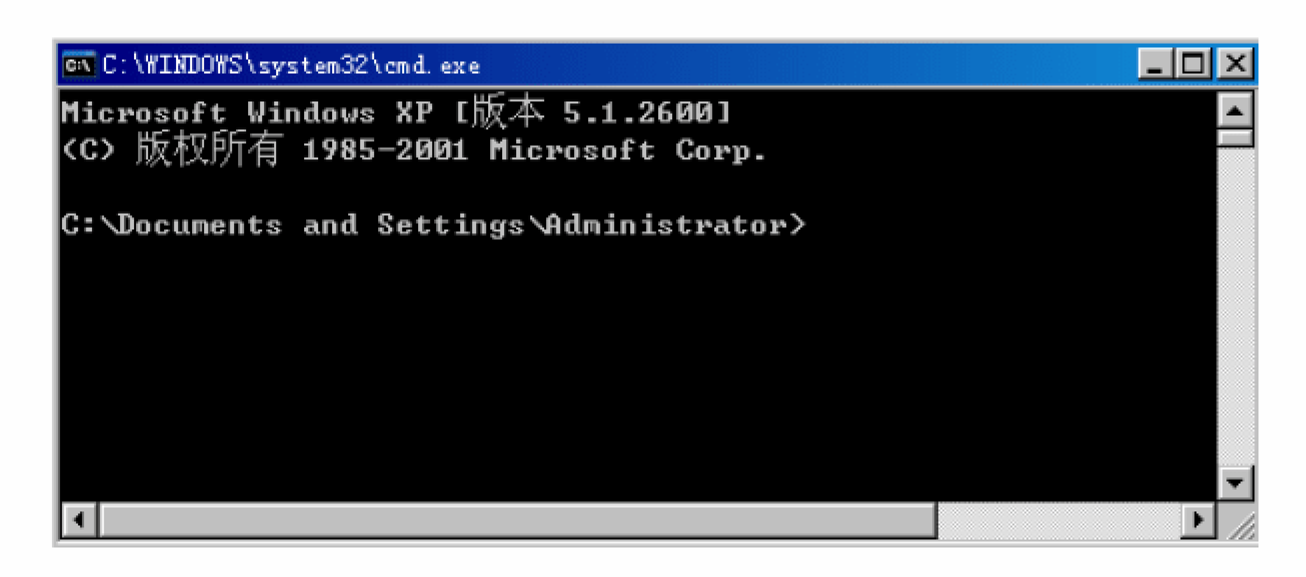


图 1-47 CMD窗口

(2) 使用如下 Android 命令创建一个 AVD。

```
android create avd --name <your_avd_name> --target <targetID>
```

其中“your_avd_name”是需要创建的 AVD 的名字，在 CMD 窗口中如图 1-48 所示。

```
C:\WINDOWS\system32\cmd.exe - android create avd --name aa --target 3
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>android create avd --name aa --target 3
Android 1.6 is a basic Android platform.
Do you wish to create a custom hardware profile [no]
```

图 1-48 创建名为“aa”的AVD

图 1-48 所示的窗口中创建了一个名为 aa，targetID 为 3 的 AVD，然后在 CMD 窗口中输入“n”，即完成操作，如图 1-49 所示。

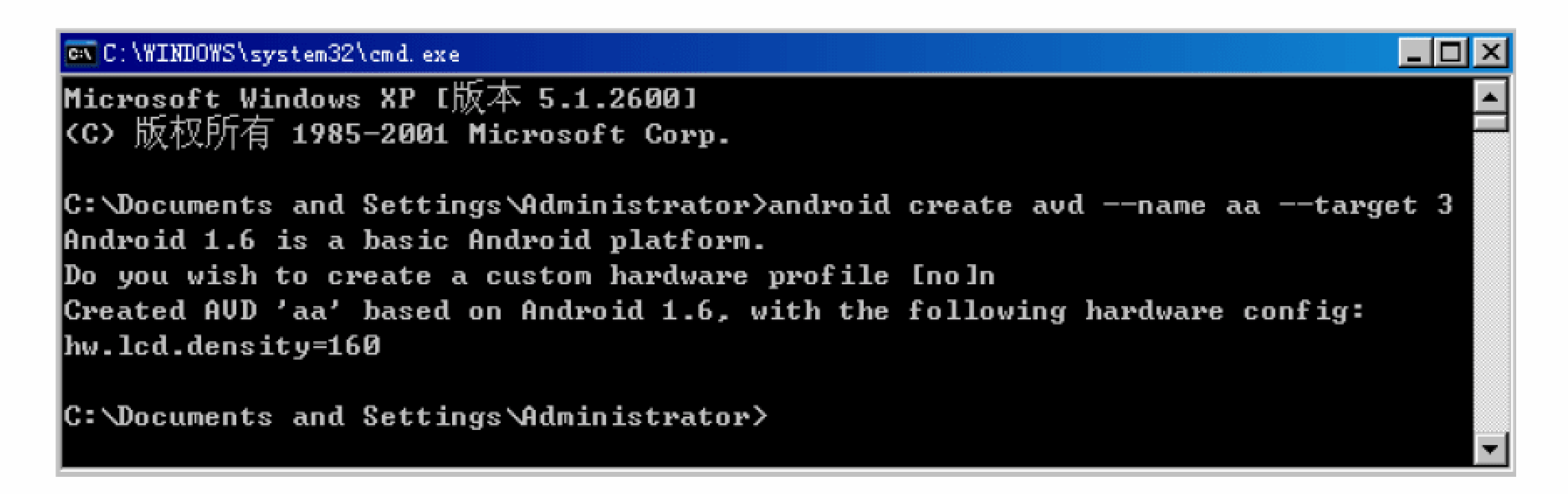


图 1-49 完成操作

Android

第2章

Android网络开发基础

Android 网络领域的范围比较广，其中主要包括网页开发、远程通信、蓝牙通信、Wi-Fi 应用、浏览器开发、流量统计和邮件处理等知识。在学习这些开发知识之前，读者需要先了解一些基础知识后才能顺利进行。从本章开始，将简要讲解开发 Android 网络项目的基础知识，为读者步入本书后面高级知识的学习打下基础。

2.1 Android 安装文件介绍

当我们下载并安装 Android SDK 后，会在安装目录中看到一些安装文件。这些文件具体是干什么用的呢？在本节的内容中，将一一为大家解开谜题。

2.1.1 Android SDK目录结构

安装 Android SDK 后，其安装目录的结构如图 2-1 所示。

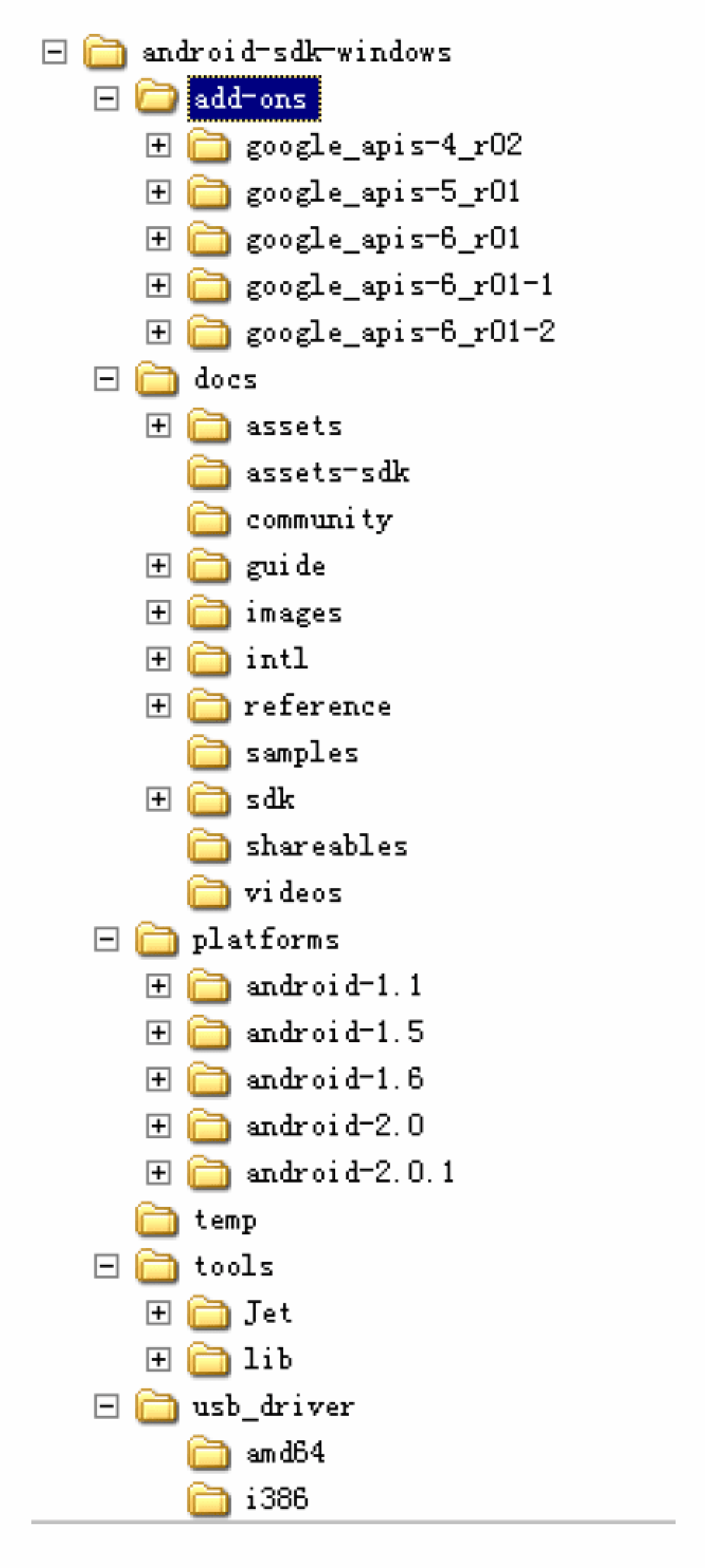


图 2-1 安装Android SDK后的目录结构

- add-ons：包含了官方提供的 API 包，最为主要的是 Map 的 API。
- docs：包含了文档，即帮助文档和说明文档。
- platforms：针对每个版本的 SDK 版本提供了与其对应的 API 包以及一些示例文

件，其中包含了各个版本的 Android，如图 2-2 所示。



图 2-2　platforms目录项

- temp：包含了一些常用的文件模板。
- tools：包含了一些通用的工具文件。
- usb_driver：包含了 AMD64 和 x86 下的驱动文件。
- SDK Setup.exe：Android 的启动文件。

2.1.2　android.jar及内部结构

在 platforms 目录下的每个 Android 版本中，都有一个名为 android.jar 的文件。例如 platforms\android-8 中的 android.jar 文件如图 2-3 所示。

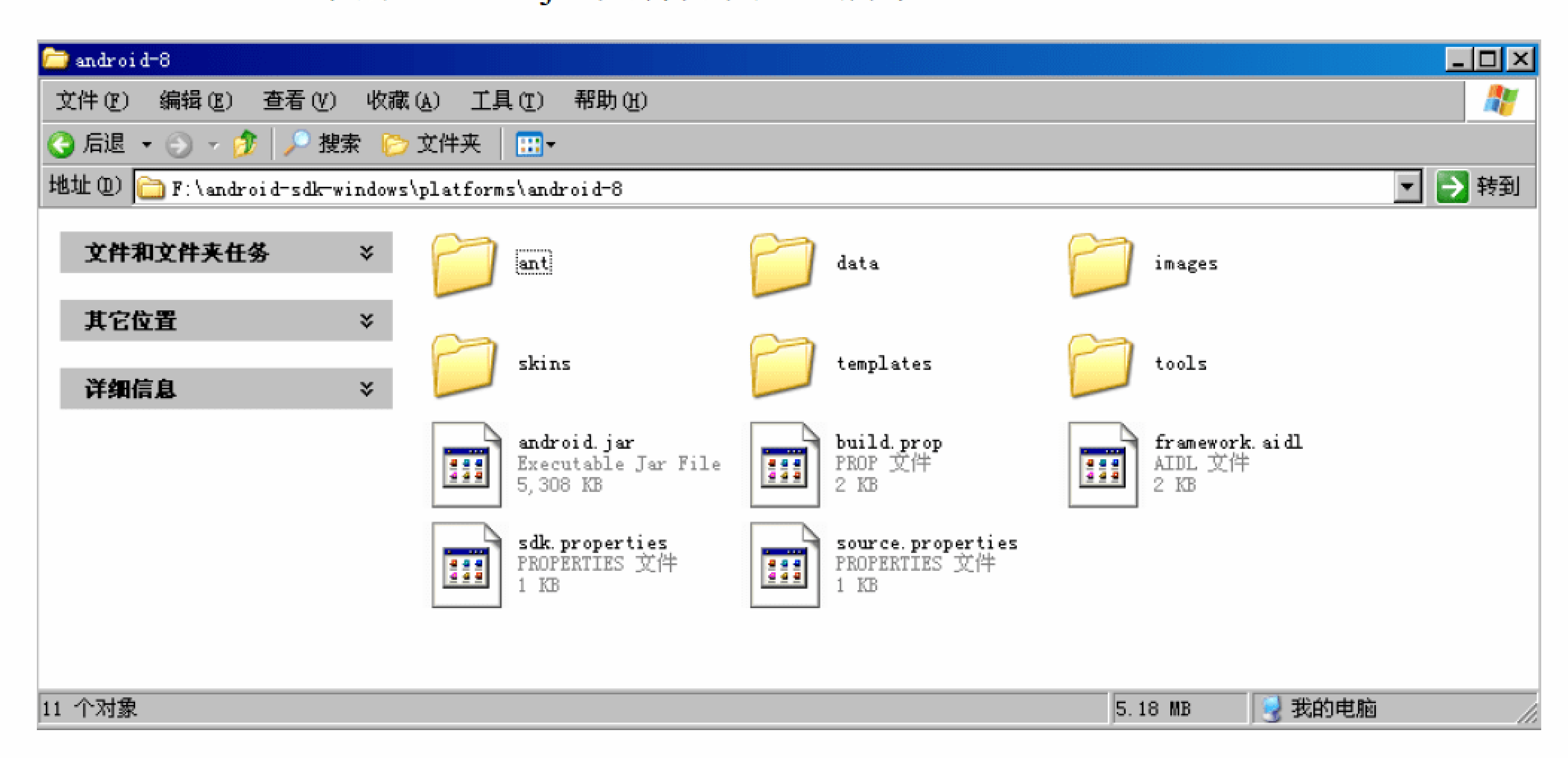


图 2-3　android.jar文件所在目录

android.jar 文件是一个标准的压缩包，其中包含了编译后的压缩文件和全部的 API。使用解压缩工具可以打开此压缩文件，解压后可以看到其内部结构如图 2-4 和图 2-5 所示。



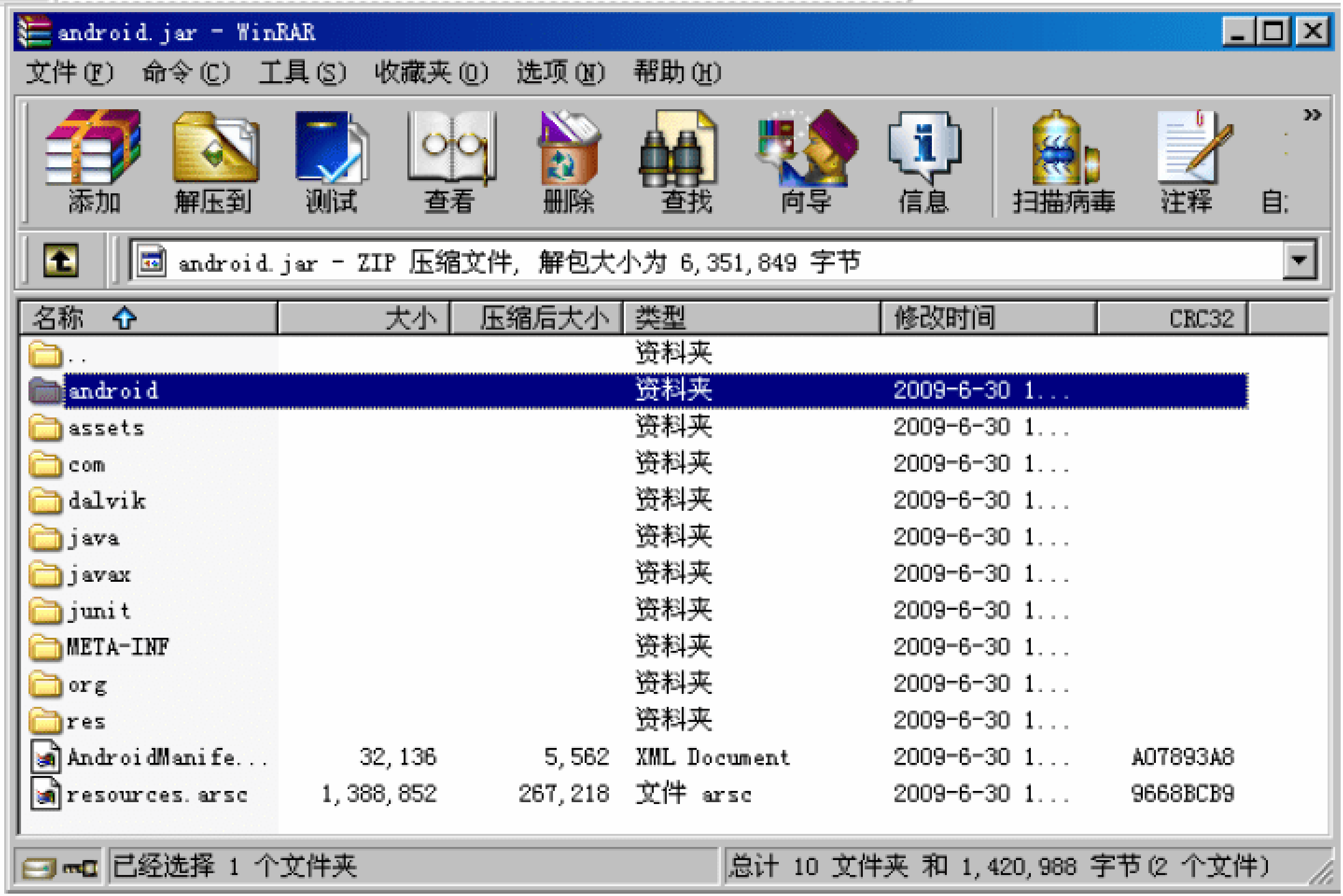


图 2-4　android.jar文件结构

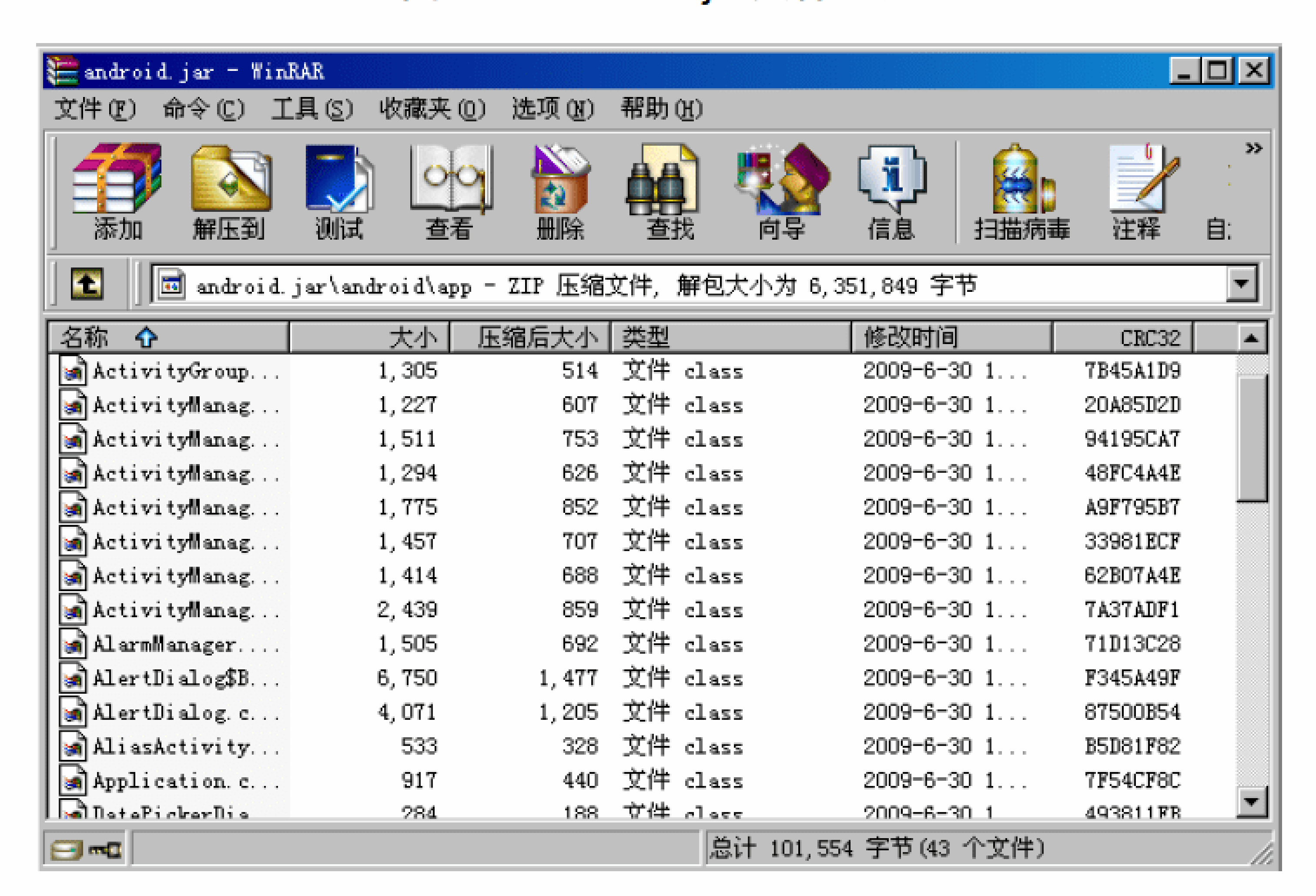


图 2-5　android.jar文件结构

2.1.3　SDK帮助文档

要想深入理解各个文件包内包含的 API 的具体用法，就必须学会阅读和查找 SDK 帮助文档。读者可以使用浏览器打开 docs 目录下的文件 index.html，如图 2-6 所示。

在图 2-6 所示的主页中，显示了 Android 基本概念和当前常用版本，在页面右侧和顶端导航中列出了一些常用的链接。此 SDK 文件对于初学者来说十分重要，可以帮助读者解决很多常见的问题，是一个很好的学习文档和帮助文档。

单击导航中的 Dev Guide 标签，打开如图 2-7 所示的页面。

图 2-7 所示的页面中，左侧是目录索引链接，单击某个链接，可以在页面的右侧显示对应的说明信息。下面我们对各个索引目录链接进行简单的介绍。

如果要想迅速地理解一个问题或知识点，可以在搜索对话框中对 SDK 进行检索，搜索到自己需要的内容。当然，很多热心的程序员和学者对 SDK 进行了翻译，网络上面世

了很多 SDK 中文版，感兴趣的读者可以从网络中获取。

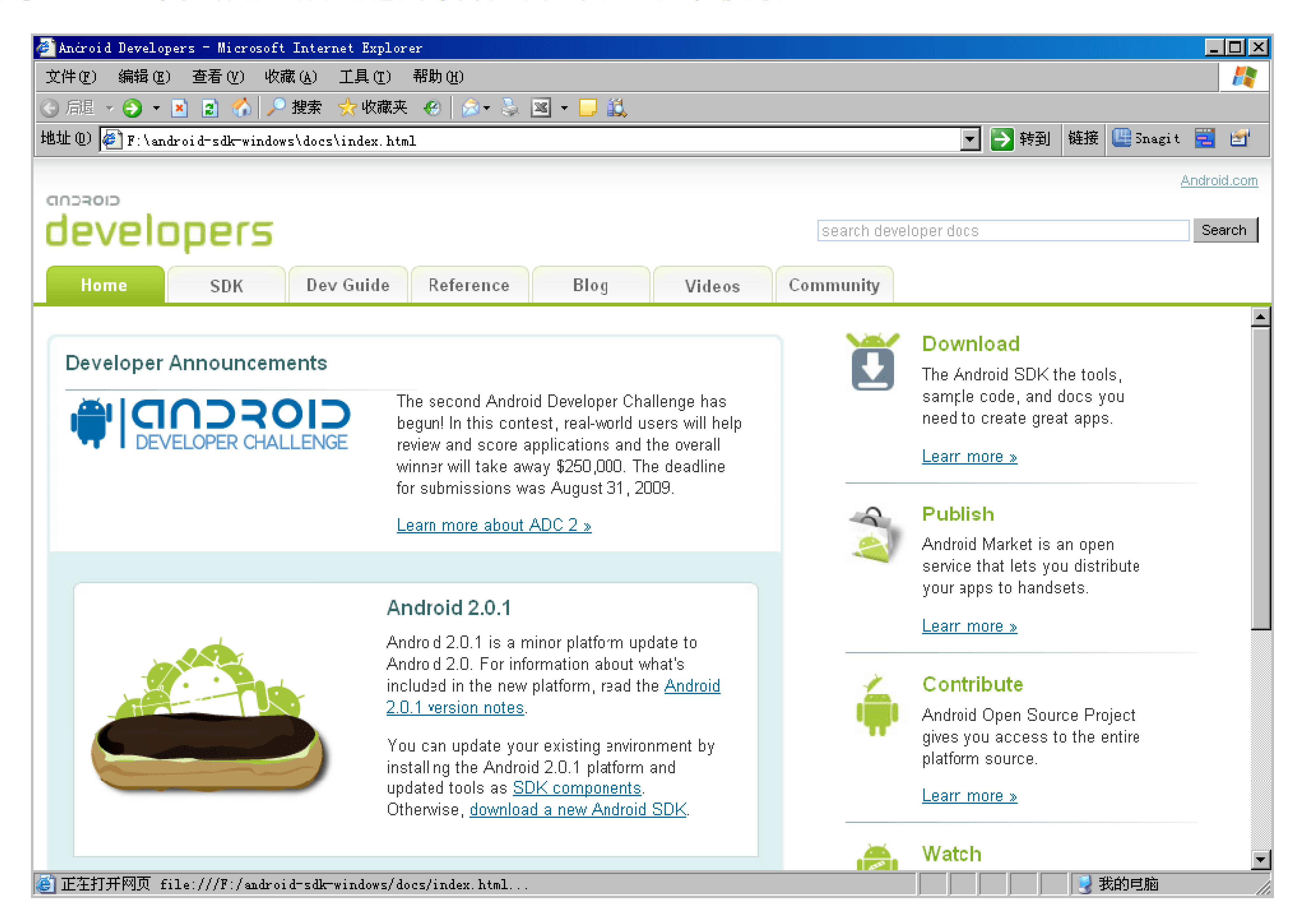


图 2-6　SDK文档主页

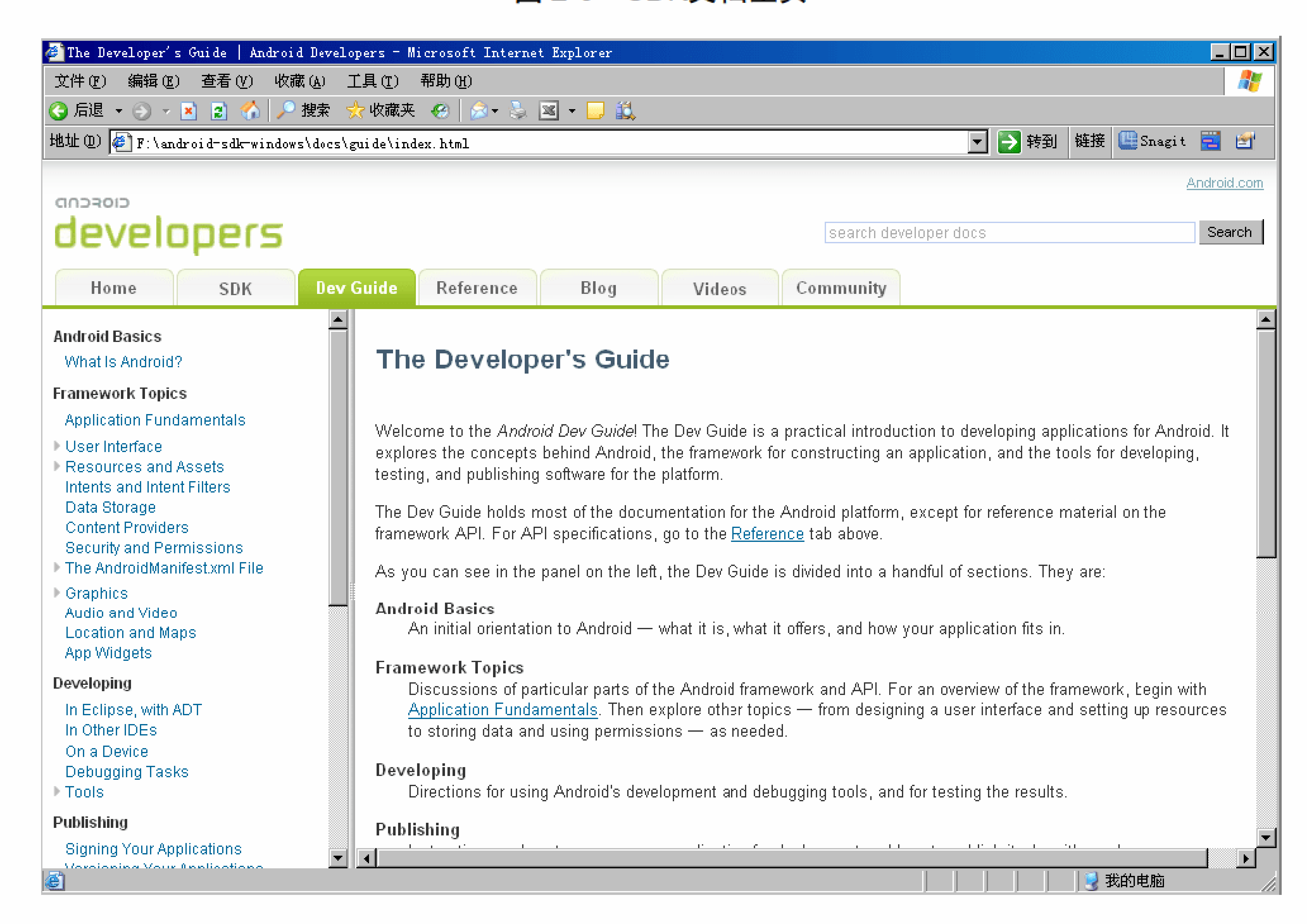


图 2-7　SDK文档目录索引

2.1.4 解析Android SDK实例

在 Android SDK 的安装目录中有一个名为 samples 的子目录，在其中保存了几个具有代表性的演示实例。这些实例从不同的方面展示了 SDK 的特性和强大功能。例如，在里面有一个名为“JetBoy”的实例，此实例是一款具备声音支持的游戏实例，它模拟演示了如何在游戏中集成 SONiVOX 的 audioINSIDE 技术的过程，此技术是 SONiVOX 捐赠给手机联盟的。此实例可以完美地播放背景音乐和场景，实现子弹击碎飞来障碍物等一系列效果。执行后的效果如图 2-8 所示。



图 2-8 JetBoy演示

2.2 分析 Android 的系统架构

本节将详细讲解 Android 应用程序的核心构成部分，为读者学习本书后面的网络应用开发打下基础。

2.2.1 Android体系结构介绍

Android 作为一个移动设备的平台，其软件层次结构包括操作系统(OS)、中间件(MiddleWare)和应用程序(Application)。根据 Android 的软件框图，其软件层次结构自下而上分为 4 层。

- 操作系统层(OS)。
- 各种库(Libraries)和 Android 运行环境(RunTime)。
- 应用程序框架(Application Framework)。
- 应用程序(Application)。

上述各个层的具体结构如图 2-9 所示。



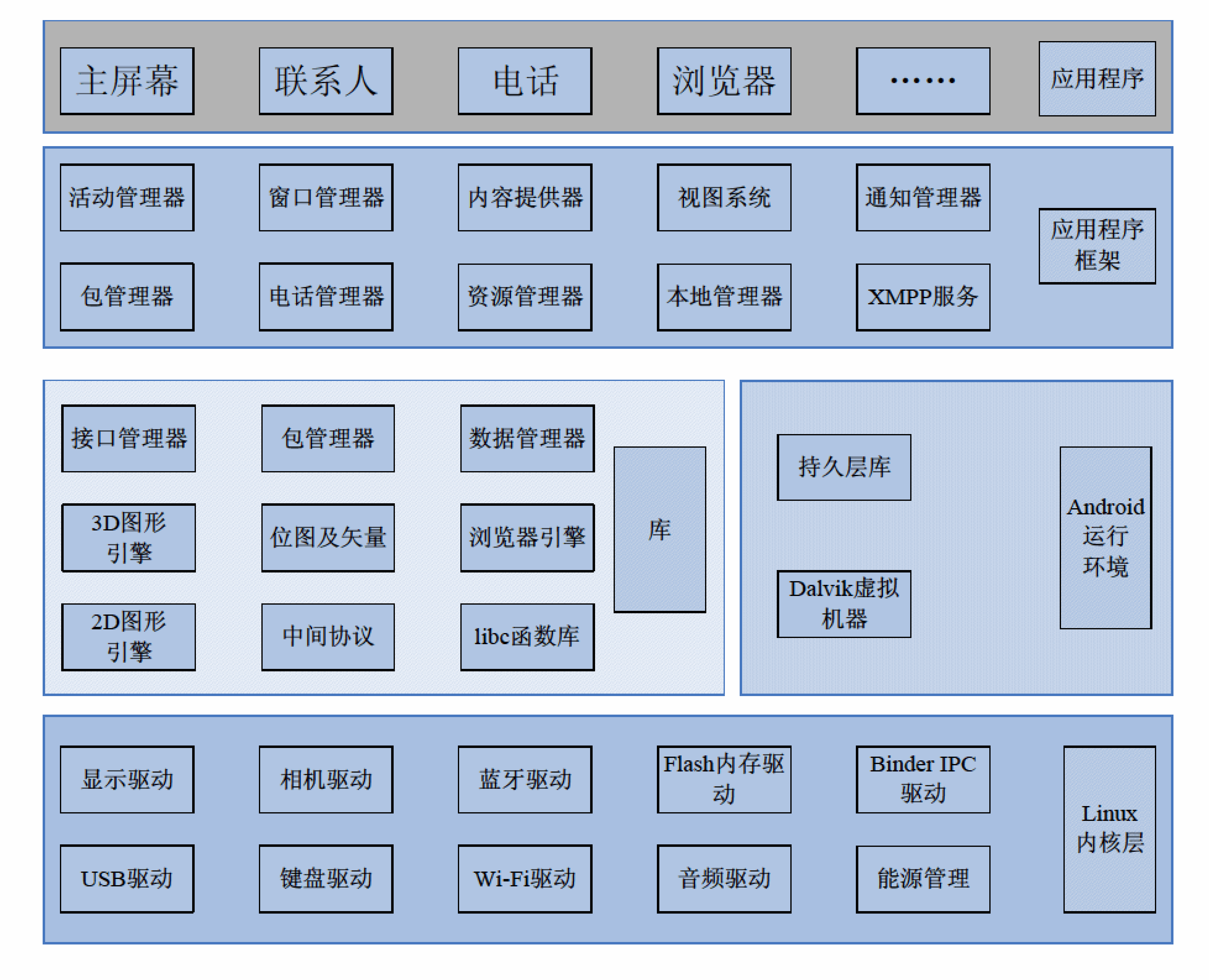


图 2-9　Android操作系统的组件结构图

1. 操作系统层(OS)

Android 使用 Linux 2.6 作为操作系统。Linux 2.6 是一种标准的技术，也是一个开放的操作系统。Android 对操作系统的使用包括核心和驱动程序两部分，Android 的 Linux 核心为标准的 Linux 2.6 内核，Android 更多的是需要一些与移动设备相关的驱动程序。主要的驱动如下。

- 显示驱动(Display Driver)：常用基于 Linux 的帧缓冲(Frame Buffer)驱动。
- Flash 内存驱动(Flash Memory Driver)：是基于 MTD 的 Flash 驱动程序。
- 相机驱动(Camera Driver)：常用基于 Linux 的 v4l(Video for Linux 的缩写)驱动。
- 音频驱动(Audio Driver)：常用基于 ALSA(Advanced Linux Sound Architecture，高级 Linux 声音体系)驱动。
- Wi-Fi 驱动：基于 IEEE 802.11 标准的驱动程序。
- 键盘驱动(KeyBoard Driver)：作为输入设备的键盘驱动。
- 蓝牙驱动(Bluetooth Driver)：基于 IEEE 802.15.1 标准的无线传输技术。
- Binder IPC 驱动：是 Android 一个特殊的驱动程序，具有单独的设备节点，提供进程间通信的功能。
- Power Management(能源管理)：管理电池电量等信息。

2. 各种库(Libraries)和Android 运行环境(RunTime)

本层次对应一般嵌入式系统，相当于中间件层次。Android 的本层次分成两个部分：

一个是各种库；另一个是 Android 运行环境。本层的内容大多是使用 C 和 C++来实现的。其中包含的各种库如下。

- C 库：C 语言的标准库，也是系统中一个最为底层的库。C 库是通过 Linux 的系统调用来实现。
- 多媒体框架(Media Framework)：这部分内容是 Android 多媒体的核心部分，基于 Packet Video(即 PV)的 OpenCORE。从功能上本库一共分为两大部分，一部分是音频、视频的回放(PlayBack)；另一部分则是音频、视频的记录(Recorder)。
- SGL：2D 图像引擎。
- SSL：即 Secure Socket Layer 位于 TCP/IP 协议与各种应用层协议之间，为数据通信提供安全支持。
- OpenGL ES 1.0：提供了对 3D 的支持。
- 界面管理工具(Surface Management)：提供了对管理显示子系统等功能。
- SQLite：一个通用的嵌入式数据库。
- WebKit：网络浏览器的核心。
- FreeType：表示位图和矢量字体的功能。

Android 的各种库一般是以系统中间件的形式提供的，它们均有一个显著的特点，就是与移动设备的平台应用密切相关。

Android 运行环境主要是指的虚拟机技术——Dalvik。Dalvik 虚拟机和一般 Java 虚拟机(Java VM)不同，它执行的不是 Java 标准的字节码(Bytecode)，而是 Dalvik 可执行格式(.dex)中的执行文件。在执行的过程中，每一个应用程序即一个进程(Linux 的一个 Process)。二者最大的区别在于 Java VM 是以基于栈(Stack-based)的虚拟机，而 Dalvik 是基于寄存器(Register-based)的虚拟机。显然，后者最大的好处在于可以根据硬件实现更大的优化，这更适合移动设备的特点。

3. 应用程序(Application)

Android 的应用程序主要是用户界面(User Interface)方面的，通常用 Java 语言编写，其中还可以包含各种资源文件(放置在 res 目录中)，Java 程序及相关资源经过编译后，将生成一个 APK 包。Android 本身提供了主屏幕(Home)、联系人(Contact)、电话(Phone)、浏览器(Browser)等众多的核心应用。同时应用程序的开发者还可以使用应用程序框架层的 API 实现自己的程序，这也是 Android 开源的巨大潜力的体现。

4. 应用程序框架(Application Framework)

Android 的应用程序框架为应用程序层的开发者提供 APIs，它实际上是一个应用程序的框架。由于上层的应用程序是以 Java 构建的，因此本层次首先包含了 UI 程序中所需要的各种控件，例如，Views(视图组件)，其中又包括 List(列表)、Grid(栅格)、Text Box(文本框)、Button(按钮)等，甚至一个嵌入式的 Web 浏览器。

一个基本的 Android 应用程序可以利用应用程序框架中的以下五个部分。

- Activity(活动)。
- Broadcast Intent Receiver(广播意图接收者)。
- Service(服务)。



- Content Provider(内容提供者)。
- Intent and Intent Filter(意图和意图过滤器)。

2.2.2　Android工程文件结构

Android 的应用工程文件主要由以下部分组成。

- src 目录：项目源文件都保存在该目录中。
- R.java 文件：该文件是 Eclipse 自动生成的，应用开发者不需要去修改其中的内容。
- Android Library：这个是应用运行的 Android 库。
- assets 目录：主要放置多媒体等一些文件。
- res 目录：主要放置应用到的资源文件。
- drawable 目录：主要放置应用到的图片资源。
- layout 目录：主要放置用到的布局文件，这些布局文件都是 XML 文件。
- values 目录：主要放置字符串(strings.xml)、颜色(colors.xml)、数组(arrays.xml)。
- AndroidManifest.xml：相当于应用的配置文件。在这个文件中，必须声明应用的名称，应用所用到的 Activity、Service 以及 receiver 等。

在 Eclipse 中，一个基本的 Android 项目的目录结构如图 2-10 所示。

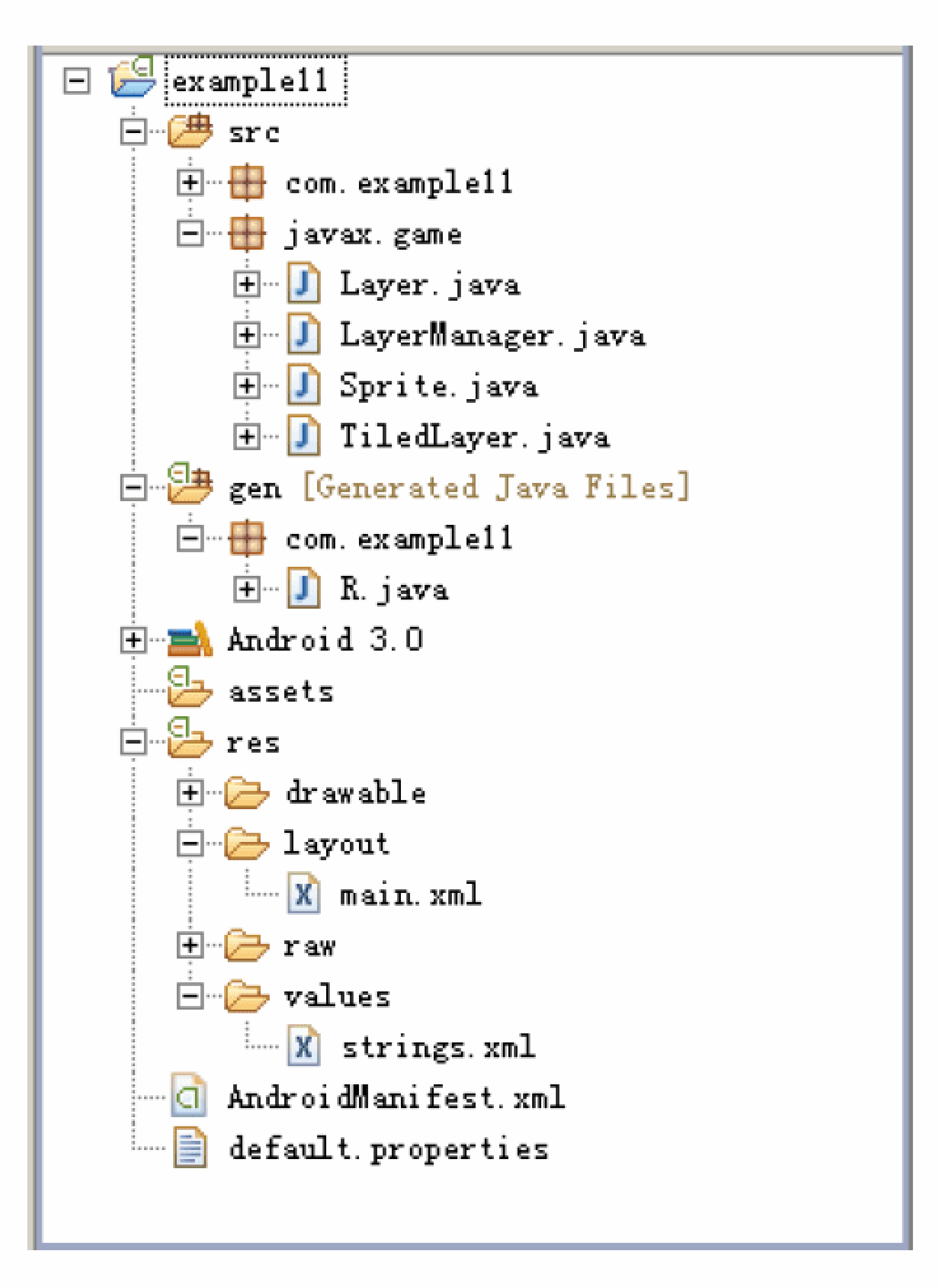


图 2-10　Android应用工程文件组成

1. src目录、res目录及R.java文件

与一般的 Java 项目一样，src 目录下保存的是项目的所有包及源文件(.java)；res 目录下包含了项目中的所有资源，例如，程序图标(drawable)、布局文件(layout)和常量(values)等。不同的是，在 Java 项目中没有 gen 目录，也没有每个 Android 项目都必须有的

AndroidManfest.xml 文件。

.java 格式文件是在建立项目时自动生成的，该文件是只读模式，不能更改。R.java 文件是定义该项目所有资源的索引文件。下面先来看看 HelloAndroid 项目的 R.java 文件，例如下面的代码：

```
package com.yarin.Android.HelloAndroid;
public final class R {
   public static final class attr {
   }
   public static final class drawable {
      public static final int icon=0x7f020000;
   }
   public static final class layout {
      public static final int main=0x7f030000;
   }
   public static final class string {
      public static final int app name=0x7f040001;
      public static final int hello=0x7f040000;
   }
}
```

从上述代码中，可以看到定义了很多常量，并且会发现这些常量的名字都与 res 目录中的文件名相同，这再次证明.java 文件中所存储的是该项目所有资源的索引。有了这个文件，在程序中使用资源将变得更加方便，可以很快地找到要使用的资源，由于这个文件不能被手动编辑，所以当我们在项目中加入了新的资源时，只需要刷新一下该项目，.java 文件便自动生成了所有资源的索引。

2. 文件AndroidManfest.xml

在文件 AndroidManfest.xml 中包含了该项目中所使用的 Activity、Service、Receiver。下面是 HelloAndroid 项目中的 AndroidManfest.xml 文件。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.yarin.Android.HelloAndroid"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon"
       android:label="@string/app name">
      <activity android:name=".HelloAndroid"
               android:label="@string/app name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
   <uses-sdk android:minSdkVersion="5" />
</manifest>
```



在上述代码中，intent-filter 描述了 Activity 启动的位置和时间。每当一个 Activity(或者操作系统)要执行一个操作时，它将创建出一个 Intent 的对象，这个 Intent 对象能承载的信息可描述你想做什么、你想处理什么数据、数据的类型，以及一些其他信息。而 Android 则会和每个 Application 所暴露的 intent-filter 的数据进行比较，找到最合适 Activity 来处理调用者所指定的数据和操作。下面来仔细分析 AndroidManfest.xml 文件，如表 2-1 所示。

表 2-1　AndroidManfest.xml分析

参　数	说　明
manifest	根节点，描述了 Package 中所有的内容
xmlns:android	包含命名空间的声明。xmlns:android=http://schemas.android.com/apk/res/android，使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据
Package	声明应用程序包
application	包含 Package 中 Application 级别组件声明的根节点。此元素也可包含 Application 的一些全局和默认的属性，如标签、icon、主题、必要的权限，等等。一个 manifest 能包含零个或一个此元素(不能大于一个)
android:icon	应用程序图标
android:label	应用程序名字
Activity	用来与用户交互的主要工具。Activity 是用户打开一个应用程序的初始页面，大部分被使用到的其他页面也由不同的 Activity 所实现，并声明在另外的 Activity 标记中。注意，每一个 Activity 必须有一个<activity>标记对应，无论它被外部使用还是只用于自己的 Package 中。如果一个 Activity 没有对应的标记，将不能运行它。另外，为了支持运行时查找 Activity，可包含一个或多个<intent-filter>元素来描述 Activity 所支持的操作
android:name	应用程序默认启动的 Activity
intent-filter	声明了指定的一组组件支持的 Intent 值，从而形成了 Intent Filter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签、icon 和其他信息
action	组件支持的 Intent action
category	组件支持的 Intent Category。这里指定了应用程序默认启动的 Activity
uses-sdk	该应用程序所使用的 SDK 版本相关

3. 常量的定义文件

下面看看资源文件中一些常量的定义，如 String.xml，例如下面的代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloAndroid!</string>
   <string name="app name">HelloAndroid</string>
</resources>
```

上述的代码非常简单，只定义了两个普通的字符串资源。

下面来分析 HelloAndroid 项目的布局文件(layout)。首先打开文件 res\layout\main.xml，其代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

在上述代码中，有以下几个布局和参数。

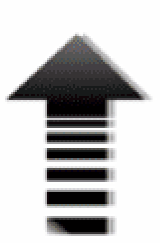
- <LinearLayout></LinearLayout>：线性版面配置，在这个标签中，所有元件都是按由上到下的顺序排成的。
- android:orientation：表示这个介质的版面配置方式是从上到下垂直地排列其内部的视图。
- android:layout_width：定义当前视图在屏幕上所占的宽度，fill_parent 即填充整个屏幕。
- android:layout_height：定义当前视图在屏幕上所占的高度，fill_parent 即填充整个屏幕。
- wrap_content：随着文字栏位的不同而改变这个视图的宽度或高度。

在上述布局代码中，使用了一个 TextView 来配置文本标签 Widget(构件)，其中设置的属性 android:layout_width 为整个屏幕的宽度，android:layout_height 可以根据文字来改变高度，而 android:text 则设置了这个 TextView 要显示的文字内容，这里引用了@string 中的 hello 字符串，即 String.xml 文件中的 hello 所代表的字符串资源。hello 字符串的内容“Hello World, HelloAndroid!”就是我们在 HelloAndroid 项目运行时看到的字符串。

注意： 上面介绍的文件只是主要文件，在项目中需要我们自行编写。在项目中还有很多其他的文件，那些文件很少需要我们自己编写，所以在此就不进行讲解了。

2.2.3 应用程序的生命周期

程序也如同自然界的生物一样，有自己的生命周期。应用程序的生命周期即程序的存活时间，即在哪段时间内有效。Android 是一个构建在 Linux 之上的开源移动开发平台，在 Android 中，多数情况下每个程序都是在各自独立的 Linux 进程中运行的。当一个程序或其某些部分被请求时，它的进程就“出生”了；当这个程序没有必要再运行下去且系统需要回收这个进程的内存用于其他程序时，这个进程就“死亡”了。可以看出，Android



程序的生命周期是由系统控制而非程序自身直接控制。这和我们编写桌面应用程序时的思维有些不同，一个桌面应用程序的进程也是在其他进程或用户请求时被创建，但是往往是在程序自身收到关闭请求后执行一个特定的动作(比如从 main 函数中返回)而导致进程结束的。要想做好某种类型的程序或者某种平台下的程序开发，最关键的就是要弄清楚这种类型的程序或整个平台下的程序的一般工作模式，并熟记在心。在 Android 中，程序的生命周期控制就属于这个范畴。

开发者必须理解不同的应用程序组件，尤其是 Activity、Service 和 Intent Receiver。了解这些组件是如何影响应用程序生命周期的，这非常重要。如果不正确地使用这些组件，可能会导致系统终止正在执行重要任务的应用程序进程。

一个常见的进程生命周期漏洞的例子是 Intent Receiver(意图接收器)，当 Intent Receiver 在 onReceive 方法中接收到一个 Intent(意图)时，它会启动一个线程，然后返回。一旦返回，系统将认为 Intent Receiver 不再处于活动状态，因而 Intent Receiver 所在的进程也就不再有用了(除非该进程中还有其他的组件处于活动状态)。因此，系统可能会在任意时刻终止该进程以回收占有的内存。这样进程中创建出的那个线程也将被终止。解决这个问题的方法是从 Intent Receiver 中启动一个服务，让系统知道进程中还有处于活动状态的工作。为了使系统能够正确决定在内存不足时应该终止哪个进程，Android 根据每个进程中运行的组件及组件的状态把进程放入一个“Importance Hierarchy(重要性分级)”中。进程的类型按重要程度如下排序。

1. 前台(Foreground)进程

前台进程与用户当前正在做的事情密切相关。不同的应用程序组件能够通过不同的方法将它的宿主进程移到前台。在如下的任何一个条件下：进程正在屏幕的最前端运行一个与用户交互的活动(Activity)，它的 onResume 方法被调用；或进程有一正在运行的 Intent Receiver(它的 IntentReceiver.onReceive 方法正在执行)；或进程有一个服务(Service)，并且在服务的某个回调函数(Service.onCreate、Service.onStart 或 Service.onDestroy)内有正在执行的代码，系统将把进程移动到前台。

2. 可见(Visible)进程

可见进程有一个可以被用户从屏幕上看到的活动，但不在前台(它的 onPause 方法被调用)。例如，如果前台的活动是一个对话框，以前的活动就隐藏在对话框之后，就会出现这种进程。可见进程非常重要，一般不允许被终止，除非是为了保证前台进程的运行而不得不终止它。

3. 服务(Service)进程

服务进程拥有一个已经用 startService 方法启动的服务。虽然用户无法直接看到这些进程，但它们做的事情却是用户所关心的，如后台 MP3 回放或后台网络数据的上传、下载。因此，系统将一直运行这些进程，除非内存不足以维持所有的前台进程和可见进程。

4. 后台(Background)进程

后台进程拥有一个当前用户看不到的活动(它的 onStop 方法被调用)。这些进程对用户

体验没有直接的影响。如果它们正确执行了活动生命周期，系统可以在任意时刻终止该进程以回收内存，并提供给前面三种类型的进程使用。系统中通常有很多这样的进程在运行，因此要将这些进程保存在 LRU 列表中，以确保当内存不足时用户最近看到的进程最后一个被终止。

5. 空(Empty)进程

空进程是不拥有任何活动的应用程序组件的进程。保留这种进程的唯一原因是在下次应用程序的某个组件需要运行时，不需要重新创建进程，这样可以提高启动速度。

系统将以进程中当前处于活动状态组件的重要程度为基础对进程进行分类。进程的优先级可能也会根据该进程与其他进程的依赖关系而提升。例如，如果进程 A 通过在进程 B 中设置 Context.BIND_AUTO_CREATE 标记或使用 ContentProvider 被绑定到一个服务(Service)，那么进程 B 在分类时至少要被看成与进程 A 同等重要。

例如 Activity 的状态转换图如图 2-11 所示。

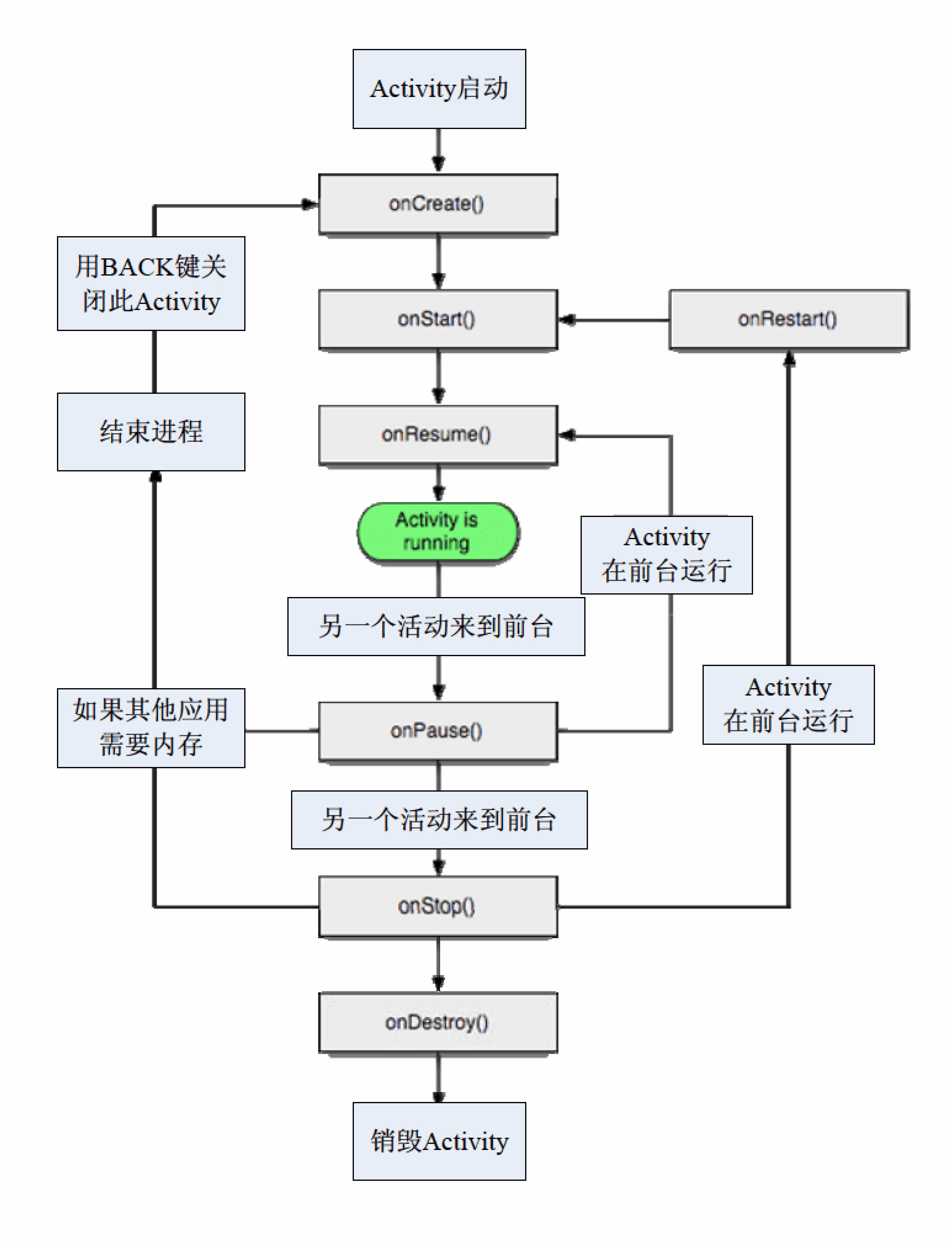


图 2-11　Activity状态转换图

图 2-11 所示的状态的变化是由 Android 内存管理器决定的，Android 会首先关闭那些



包含 Inactive Activity 的应用程序，然后关闭 Stopped(已停止)状态的程序。在极端情况下，会移除 Paused(暂停)状态的程序。

2.3　网页开发基础

在作者写作本书时，移动上网设备已经超越电脑。由此可以推断出，有众多的用户使用手机等移动设备进行网上冲浪。所以对于 Android 网络应用程序员来说，很有必要掌握在 Android 手机中开发网页的知识。

2.3.1　HTML简介

HTML 是由标记组成的一种网页标记语言，当前几乎所有的网页都是通过 HTML 展现在我们眼前的。目前最新的 HTML 版本是刚刚推出的 HTML 5。

1. HTML基本结构

HTML 是一种网页标记语言，它的所有部分都是标记<>和</>括起来。来看下面的代码：

```
<html>
<head>
<title>这是网页的标题标签</title>
</head>
<body>
这是网页内容
<body>
</html>
```

上面展示的代码，其实就是一个很简单的网页，网页就是通过这种方式展现给浏览者，其中各个参数介绍如下。

- <html>…</html>：这是 HTMl 标签，所有标记都要放在这里，<html>是开始标签，</html>是标签的结束。
- <head>…</head>：表示网页的头部。
- <title>…</ title >：表示网页的标题。
- <body>…</body>：表示网页的内容。

2. HTML标记特性

HTML 必须以<html>开始，以</html>结束，文件头包含在<head>…</head>里面，文件体包含在<body>…</body>里面，在文件头部，用户可以用<title>…</title>标记来声明文件标题。在 HTML 文档中，值得提醒读者的是 HTML 也有注释，它和 Java 是完全不同的，HTML 采用<!—注释-->来标记注释，在 HTML 中，每一个标记都是成对出现。下面展示一段代码。

```
<html>
<head>
<title>欢迎进入 Java 网络世界</title>
</head>
<body>
这里是 Java 网络世界！
<body>
</html>
```

将上述文件以.html 扩展名保存，双击打开后会得到如图 2-12 所示的效果。



图 2-12　用HTML实现的一个网页

注意：HTML 技术虽然比较简单，但它是我们开发网页的基础。因为 HTML 不是本书的重点，如果读者对 HTML 技术不是很熟悉，建议参考相关图书和网络资料来学习掌握。

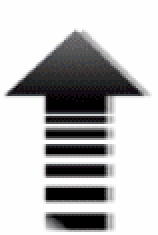
2.3.2　XML技术

XML是eXtensible Markup Language的缩写，表示可扩展标记语言。XML与HTML一样，都是SGML(Standard Generalized Markup Language，标准通用标记语言)。XML是Internet环境中跨平台的、依赖于内容的技术，是当前处理结构化文档信息的有力工具。扩展标记语言XML是一种简单的数据存储语言，使用一系列简单的标记描述数据，而这些标记可以用方便的方式建立。虽然XML比二进制数据要占用更多的空间，但XML极其简单，易于掌握和使用。

1. XML的概述

XML与Access、Oracle和SQL Server等专业数据库不同，这些专业数据库提供了更强有力的数据存储和分析能力，例如，数据索引、排序、查找、相关一致性等，而XML仅仅实现了展示数据的功能。事实上XML与其他数据表现形式最大的不同是极其简单。但正是这点使XML与众不同。

XML 的简单使其易于在任何应用程序中读写数据，这使 XML 很快成为数据交换的唯一公共语言，虽然不同的应用软件也支持其他的数据交换格式，但不久之后它们都将支持XML，那就意味着程序可以更容易地与 Windows、Mac OS、Linux 以及其他平台下产生的信息结合，然后可以很容易加载 XML 数据到程序中并进行分析，并以 XML 格式输出结果。



为了使得 SGML 显得用户友好，XML 重新定义了 SGML 的一些内部值和参数，去掉了大量的很少用到的功能，这些繁杂的功能使得 SGML 在设计网站时显得复杂化。XML 保留了 SGML 的结构化功能，这样就使得网站设计者可以定义自己的文档类型，XML 同时也推出一种新型文档类型，使得开发者也可以不必定义文档类型。

因为XML是W3C制定的，XML的标准化工作由W3C的XML工作组负责，该小组成员由来自各个地方和行业的专家组成，他们通过Email交流对XML标准的意见，并提出自己的看法(www.w3.org/TR/WD-xml)。因为XML 是个公共格式，它不专属于任何一家公司，所以不必担心XML技术会成为少数公司的盈利工具。

2. XML的语法

上面虽然讲解了 XML 的特点，但是初学者仍然不明白 XML 是用来做什么的。其实 XML 什么也不做，它只是用来存储数据的，对 HTML 语言进行扩展。它和 HTML 分工很明显，XML 是用来存储数据，而 HTML 是用来如何表现数据的。下面通过一段演示代码进行讲解。

```
<?xml version="1.0" encoding="utf-8"?>
<book>
<person>
<first>Kiran</first>
<last>Pai</last>
<age>22</age>
</person>
<person>
<first>Bill</first>
<last>Gates</last>
<age>46</age>
</person>
<person>
<first>Steve</first>
<last>Jobs</last>
<age>40</age>
</person>
</book>
```

上面的语法不但可以这么写，只要符合语法还可以写成汉语，如下面的代码(3-3.xml)：

```
<?xml version="1.0" encoding="utf-8"?>
     <项目>
         <名>天上星</名>
         <电子邮件>tianshangxing@hotmail.com</电子邮件>
         <住宅>何国何市何区何街道何番号</住宅>
         <电话>86-021-742745674</电话>
         <一言>XML 学习</一言>
</项目>
```

从上面两段代码可以看出，XML 的标记完全自由定义，不受约束，它只是用来存储信息。除了第一行固定以外，其他的只需要前后标签一致，不能省掉末标签。下面是对

XML 语法格式的总结。

- 第一行要对 XML 进行声明，也就是 XML 的版本。
- XML 的标记和 HTML 一样是成双成对出现的。
- XML 对标记的大小写十分敏感。
- XML 标记是用户自行定义，但是每一个标记必须有结束标记。

注意：XML 技术虽然比较简单，但是我们开发网页的基础。因为 XML 不是本书的重点，如果读者对 XML 技术不是很熟悉，建议参考相关图书和网络资料来学习掌握。

2.3.3 CSS技术

CSS 技术是 Web 网页技术的重要组成，页面通过 CSS 的修饰可以实现用户需要的显示效果。因为在现实应用中经常用到的 CSS 元素是选择符、属性和值，所以在 CSS 的应用语法中其主要应用格式也主要涉及上述 3 种元素。CSS 的基本语法结构如下：

```
<style type="text/css">
<!--
.选择符{属性：值}
-->
</style>
```

其中，CSS 选择符的种类有多种，并且命名机制也不相同。例如下面的代码演示了 CSS 技术在网页中的作用。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>无标题文档</title>
<style type="text/css">                                    <!--设置的样式-->
<!--
.mm {
    font-family: "Times New Roman", Times, serif;          /*设置字体*/
    font-size: 18px;                                       /*设置字体大小*/
    font-weight: bold;                                     /*加粗字体*/
    color: #990000;                                        /*设置颜色*/
}
-->
</style>
</head>
<body class="mm">                                          <!--文本调用样式-->
我的未来不是梦
</body>
</html>
```

执行后的效果如图 2-13 所示。



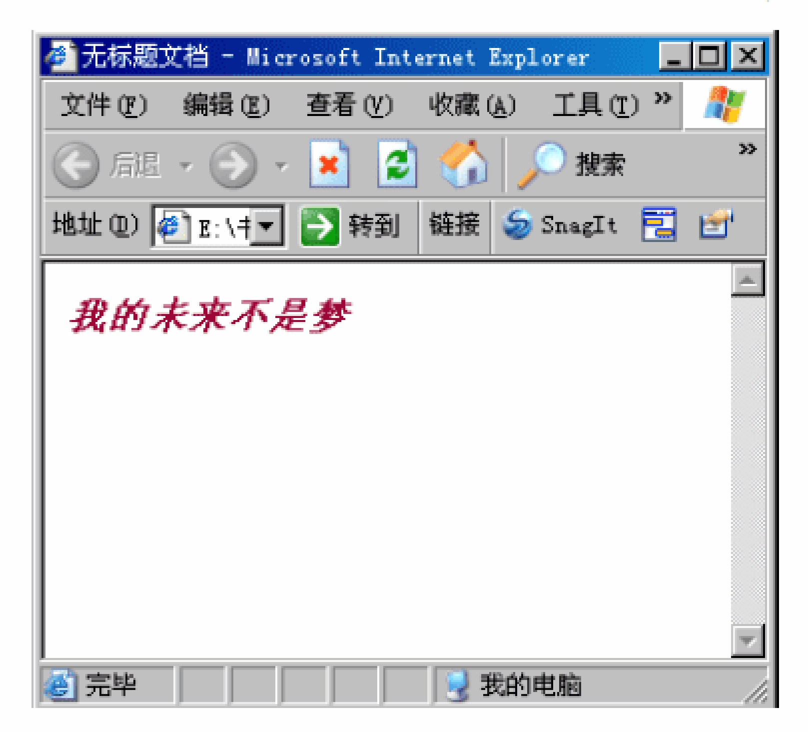


图 2-13　执行效果

注意：CSS 技术虽然比较简单，但是我们开发网页的基础。因为 CSS 不是本书的重点，如果读者对 CSS 技术不是很熟悉，建议参考相关图书和网络资料来学习掌握。

2.3.4　JavaScript技术

JavaScript 是一种脚本技术，页面通过脚本程序可以实现用户数据的传输和动态交互。JavaScript 是一种基于对象(Object)和事件驱动(Event Driven)并具有安全性能的脚本语言。其目的是与 HTML 超文本标记语言、Java 脚本语言(Java 小程序)相互结合，实现 Web 页面中链接多个对象，并与 Web 客户交互的效果，从而实现客户端应用程序的开发。

使用 JavaScript 技术的基本语法格式如下：

```
<Script Language ="JavaScript">
JavaScript 脚本代码 1
JavaScript 脚本代码 2
…
</Script>
```

2.4　简析 Android 内核

虽然本书的主要内容是讲解 Android 网络应用开发的知识，但是为了让读者更加深入地了解每一个网络领域的具体原理，需要讲解一些底层和内核方面的知识作为补充和铺垫，所以下面讲解一些 Android 内核源码和驱动开发方面的知识。

2.4.1　Android继承于Linux

Android 是在 Linux 2.6 的内核基础之上运行的，提供的核心系统服务包括安全、内存管理、进程管理、网络组和驱动模型等内容。内核部分还相当于一个介于硬件层和系统中其他软件组之间的抽象层次。但是严格来说它不算是 Linux 操作系统。

因为 Android 内核是由标准的 Linux 内核修改而来的，所以它继承了 Linux 内核的诸

多优点，保留了 Linux 内核的主体架构。同时 Android 按照移动设备的需求，在文件系统、内存管理、进程间通信机制和电源管理等方面进行了修改，添加了相关的驱动程序和必要的新功能。但是和其他精简的 Linux 系统相比，例如 uClinux，Android 很大程度上保留了 Linux 的基本架构，因此 Android 的应用性和扩展性更强。

2.4.2　Android内核和Linux内核的区别

Android 系统的系统层面的底层是 Linux，并且在中间加上了一个叫作 Dalvik 的 Java 虚拟机，这从表面层看是 Android 运行库。每个 Android 应用都运行在自己的进程上，享有 Dalvik 虚拟机为其分配的专有实例。为了支持多个虚拟机在同一个设备上高效运行，Dalvik 被改写过。

Dalvik 虚拟机执行的是 Dalvik 格式的可执行文件(.dex)。该格式经过优化，以降低内存耗用到最低。Java 编译器将 Java 源文件转换为 class 文件，class 文件又被内置的 dx 工具转化为 dex 格式文件，这种文件在 Dalvik 虚拟机上注册并运行。

Android 系统的应用软件都是运行在 Dalvik 之上的 Java 软件，而 Dalvik 是运行在 Linux 中的，在一些底层功能，比如线程和低内存管理方面，Dalvik 虚拟机是依赖 Linux 内核的。由此可见，Android 是运行在 Linux 之上的操作系统，但是它本身不能算是 Linux 的某个版本。

Android 内核和 Linux 内核的差别主要体现在以下 11 个方面。

1) Android Binder

Android Binder 是基于 OpenBinder 框架的一个驱动，用于提供 Android 平台的进程间通信(IPC，Inter-Process Communication)。原来的 Linux 系统上层应用的进程间通信主要是 D-bus(desktop bus)，采用消息总线的方式来进行 IPC。

其源代码位于：drivers/staging/android/binder.c。

2) Android 电源管理(PM)

Android 电源管理是一个基于标准 Linux 电源管理系统的轻量级的 Android 电源管理驱动，针对嵌入式设备做了很多优化。利用锁和定时器来切换系统状态，控制设备在不同状态下的功耗，以达到节能的目的。

其源代码分别位于以下文件：

- kernel/power/earlysuspend.c
- kernel/power/consoleearlysuspend.c
- kernel/power/fbearlysuspend.c
- kernel/power/wakelock.c
- kernel/power/userwakelock.c

3) 低内存管理器(Low Memory Killer)

Android 中的低内存管理器和 Linux 标准的 OOM(Out Of Memory)相比，其机制更加灵活，它可以根据需要中止进程来释放需要的内存。Low Memory Killer 的代码非常简单，里面的关键是函数 Lowmem_shrinker()。作为一个模块在初始化时调用 register_shrinke 注册 Lowmem_shrinker，它会被 vm 在内存紧张的情况下调用。Lowmem_shrinker 完成具体操



作，简单说就是寻找一个最合适的进程杀死，从而释放它占用的内存。

其源代码位于：drivers/staging/android/lowmemorykiller.c

4) 匿名共享内存(Ashmem)

匿名共享内存为进程间提供大块共享内存，同时为内核提供回收和管理这个内存的机制。如果一个程序尝试访问 Kernel 释放的一个共享内存块，它将会收到一个错误提示，然后重新分配内存并重载数据。

其源代码位于：mm/ashmem.c

5) Android PMEM(Physical)

PMEM 用于向用户空间提供连续的物理内存区域，DSP 和某些设备只能工作在连续的物理内存上。驱动中提供了 mmap、open、release 和 ioctl 等接口。

源代码位于：drivers/misc/pmem.c

6) Android Logger

Android Logger 是一个轻量级的日志设备，用于抓取 Android 系统的各种日志，是 Linux 所没有的。

其源代码位于：drivers/staging/android/logger.c

7) Android Alarm

Android Alarm 提供了一个定时器用于把设备从睡眠状态唤醒，同时它也提供了一个即使在设备睡眠时也会运行的时钟基准。

其源代码位于以下文件：

- drivers/rtc/alarm.c
- drivers/rtc/alarm-dev.c

8) USB Gadget 驱动

此驱动是一个基于标准 Linux USB Gadget 驱动框架的设备驱动，Android 的 USB 驱动是基于 Gadget 框架的。

其源代码位于以下文件：

- drivers/usb/gadget/android.c
- drivers/usb/gadget/f_adb.c
- drivers/usb/gadget/f_mass_storage.c

9) Android RAM Console

为了提供调试功能，Android 允许将调试日志信息写入一个被称为 RAM Console 的设备中，它是一个基于 RAM 的 Buffer。

其源代码位于：drivers/staging/android/ram_console.c

10) Android timed device

Android timed device 提供了对设备进行定时控制的功能，目前仅仅支持 vibrator 和 LED 设备。

其源代码位于：drivers/staging/android/timed_output.c(timed_gpio.c)

11) Yaffs2 文件系统

在 Android 系统中，采用 Yaffs2 作为 MTD NAND Flash 文件系统。Yaffs2 是一个快速稳定的、应用于 NAND 和 NOR Flash 的跨平台嵌入式设备文件系统，同其他 Flash 文件系

统相比，Yaffs2 使用更小的内存来保存它的运行状态，因此占用内存小；Yaffs2 的垃圾回收非常简单而且快速，因此能达到更好的性能；Yaffs2 在大容量的 NAND Flash 上性能表现尤为明显，非常适合大容量的 Flash 存储。

其源代码位于：fs/yaffs2/目录。

2.5 简要分析 Android 源码

源码分析是深入掌握 Android 网络应用知识的前提工作，在本节的内容中，将讲解分析 Android 源码的基本知识，为读者步入本书后面知识的学习打下基础。

2.5.1 获取并编译Android源码

在分析 Android 源码之前，需要先下载获取 Android 源码。读者可以登录 http://source.android.com/ 获取 Android 的源码，在网页 http://source.android.com/source/downloading.html 中详细介绍了获取 Android 源码的方法，如图 2-14 所示。

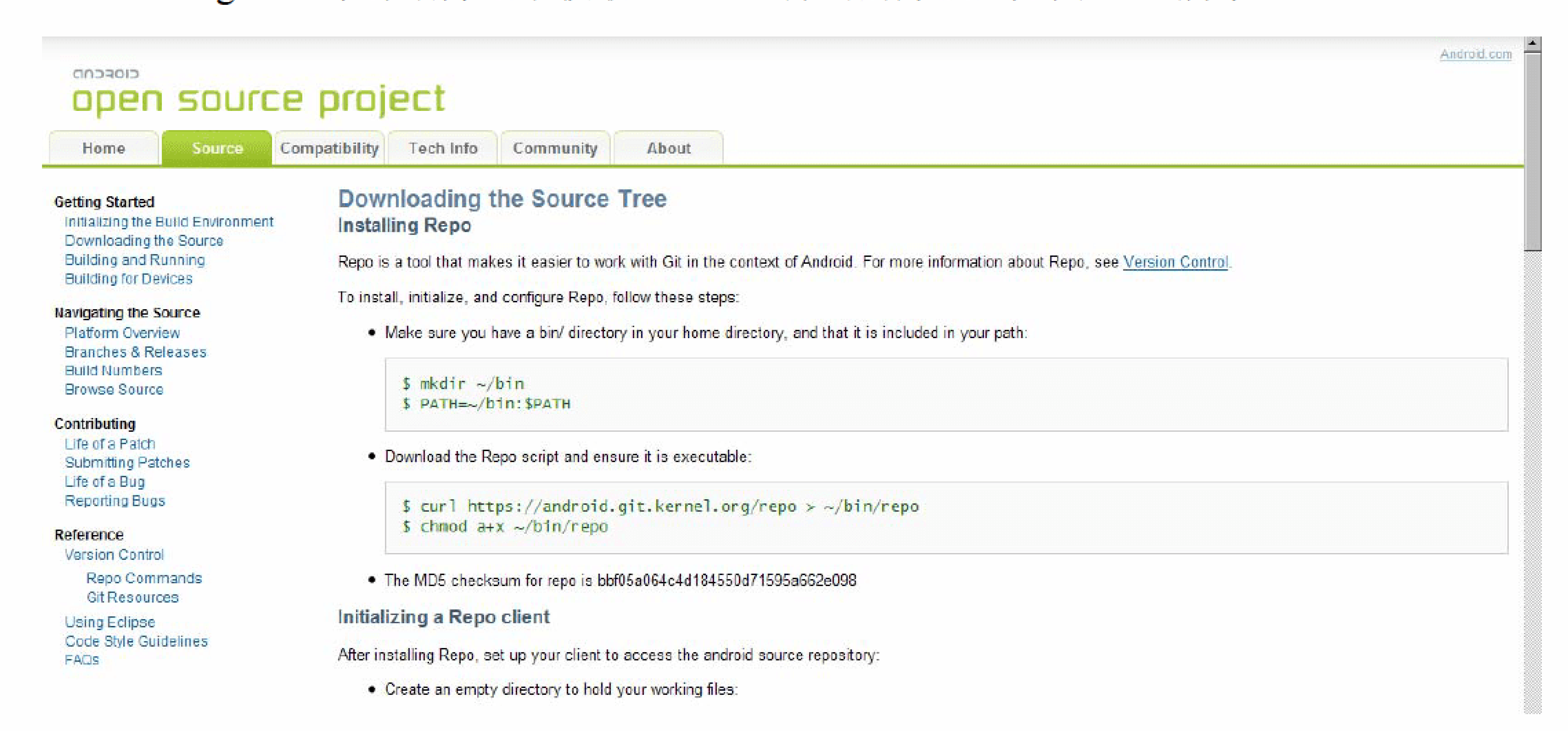


图 2-14　Linux下获取Android源码的方法

在下载源码时，需要使用 repo 或 git 工具来实现。下面将详细介绍使用工具获取 Android 源码的流程。

(1) 创建源代码下载目录，命令如下：

```
mkdir /work/android-froyo-r2
```

(2) 用 repo 工具初始化一个版本，假如是 Android 2.2r2，则命令如下：

```
cd /work/android-froyo-r2
repo init -u git://android.git.kernel.org/platform/manifest.git -b froyo
```

在初始化过程中会显示相关的版本 TAG 信息，同时会提示我们输入用户名和邮箱地



址，上面的命令初始化的是 Android 2.2 froyo 的最新版本。

(3) 因为 Android 2.2 有很多个版本，这些版本信息可以从 TAG 信息中看出来。目前 froyo 的所有版本信息如下：

```
* [new tag]        android-2.2.1 r1 -> android-2.2.1 r1
* [new tag]        android-2.2 r1 -> android-2.2 r1
* [new tag]        android-2.2 r1.1 -> android-2.2 r1.1
* [new tag]        android-2.2 r1.2 -> android-2.2 r1.2
* [new tag]        android-2.2 r1.3 -> android-2.2 r1.3
* [new tag]        android-cts-2.2 r1 -> android-cts-2.2 r1
* [new tag]        android-cts-2.2 r2 -> android-cts-2.2 r2
* [new tag]        android-cts-2.2_r3 -> android-cts-2.2_r3
```

每次下载的都是最新的版本，当然也可以根据 TAG 信息下载某一特定的版本。例如下面的命令：

```
repo init -u git://android.git.kernel.org/platform/manifest.git -b
android-cts-2.3_r1
```

(4) 开始下载源码，命令如下：

```
repo sync
```

froyo 版本的代码很大，超过 2GB，下载过程非常漫长，需要读者耐心等待。

(5) 编译代码，命令如下：

```
cd /work/android-froyo-r2
make
```

上述 repo 获取的方式速度非常慢，我们可以通过网页浏览的方式来访问 Android 代码库，其浏览路径是 http://android.git.kernel.org/，界面如图 2-15 所示。

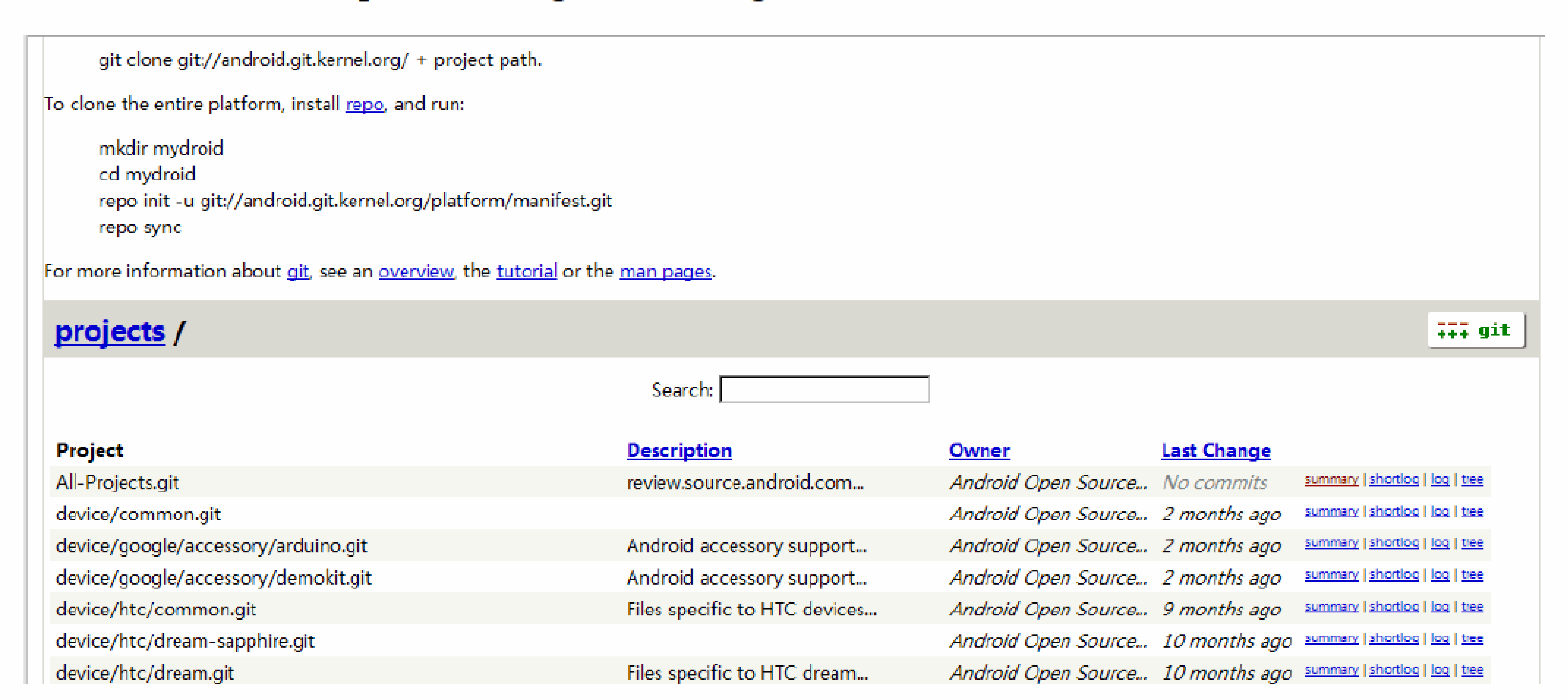


图 2-15　页面浏览方式访问Android代码库

注意： 因为上述获取 Android 源码的过程非常缓慢，所以一般建议不要使用 repo 来下载 Android 源码，而直接登录 http://www.androidin.com/bbs/pub/cupcake.tar.gz 来下载，解压出来的 cupcake 中也有.repo 文件夹，此时可以通

过 repo sync 来更新 cupcake 代码。命令如下：

```
tar -xvf  cupcake.tar.gz
```

2.5.2　Android对Linux的改造

Android 内核是基于 Linux 2.6 内核的，是一个增强的内核版本，除了修改部分 Bug 外，还提供了用于支持 Android 平台的设备驱动。Android 不但使用了 Linux 内核的基本功能，而且对 Linux 进行了改造，以实现更为强大的通信功能。

Android 中的 Linux 内核与驱动结构如图 2-16 所示。

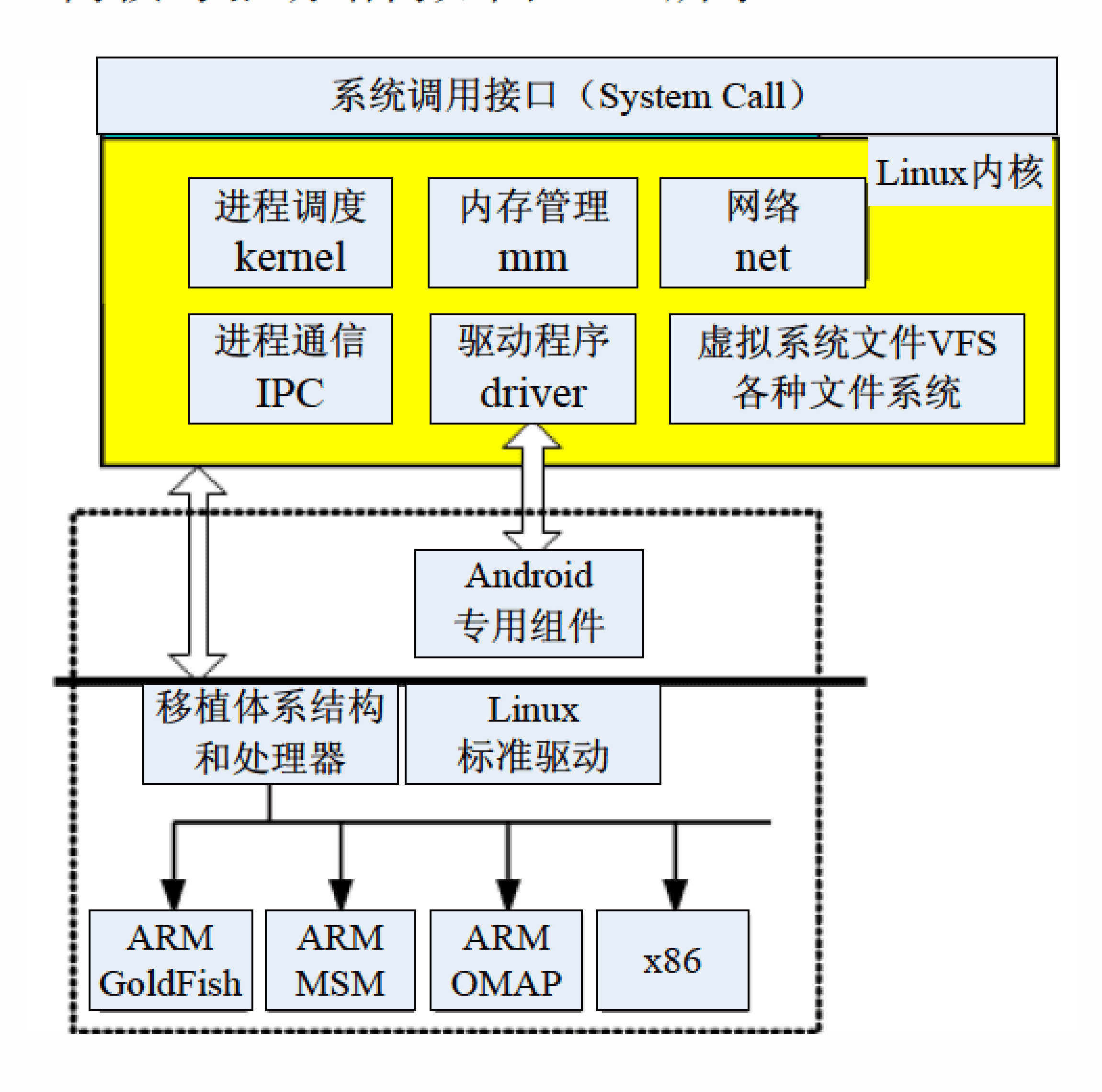


图 2-16　Android中的Linux内核与驱动结构

2.5.3　为Android构建Linux的操作系统

如果我们以一个原始的 Linux 操作系统为基础，改造成为一个适合于 Android 的系统，所做的工作其实非常简单，仅仅是增加适用于 Android 的驱动程序。在 Android 中有很多 Linux 系统的驱动程序，将这些驱动程序移植到新系统的步骤非常简单，具体来说有以下三个步骤。

(1) 编写新的源代码。

(2) 在 KConfig 配置文件中增加新内容。

(3) 在 Makefile 中增加新内容。

在 Android 系统中，通常会使用 FrameBuffer 驱动、Event 驱动、Flash MTD 驱动、Wi-Fi 驱动、蓝牙驱动和串口等驱动程序。并且还需要音频、视频、传感器等驱动和 sysfs 接口。移植的过程就是移植上述驱动的过程，我们的工作是在 Linux 下开发适用于 Android 的驱动程序，并移植到 Android 系统。

在 Android 中添加扩展驱动程序的基本步骤如下。

(1) 在 Linux 内核中移植硬件驱动程序，实现系统调用接口。



(2) 把硬件驱动程序的调用在 HAL 中封装成 Stub。

(3) 为上层应用的服务实现本地库，由 Dalvik 虚拟机调用本地库来完成上层 Java 代码的实现。

(4) 编写 Android 应用程序，提供 Android 应用服务和用户操作界面。

2.6　总结和网络应用有关的包

在 Android 系统中开发网络项目时，需要用到 Android SDK 中为我们提供的包，这些包的具体说明如表 2-2 所示。

表 2-2　Android SDK中和网络有关的包

包	说　明
java.net	提供与联网有关的类，包括流和数据包(datagram)sockets、Internet 协议和常见 HTTP 处理。该包是一个多功能网络资源。有经验的 Java 开发人员可以立即使用这个熟悉的包创建应用程序
java.io	虽然没有提供显式的联网功能，但是仍然非常重要。该包中的类由其他 Java 包中提供的 socket 和连接使用。它们还用于与本地文件(在与网络进行交互时会经常出现)的交互
java.nio	包含表示特定数据类型的缓冲区的类。适用于两个基于 Java 语言端点之间的通信
org.apache.*	表示许多为 HTTP 通信提供精确控制和功能的包。可以将 Apache 视为流行的开源 Web 服务器
android.net	除核心 java.net.* 类以外，包含额外的网络访问 socket。该包包括 URI 类，后者频繁用于 Android 应用程序开发，而不仅仅是传统的联网方面
android.net.http	包含处理 SSL 证书的类
android.net.wifi	包含在 Android 平台上管理有关 Wi-Fi(802.11 无线 Ethernet)所有方面的类。并不是所有设备都配备了 Wi-Fi 功能，特别是 Android 在 Motorola 和 LG 等手机制造商的“翻盖手机”领域获得了成功
android.telephony.gsm	包含用于管理和发送 SMS(文本)消息的类。一段时间后，可能会引入额外的包来为非 GSM 网络提供类似的功能，比如 CDMA 或 android.telephony.cdma 等网络

Android

第 3 章

HTTP通信处理

在 Android 网络应用中，通常是用 HTTP 协议来传输网络数据。在 Android 平台中开发 HTTP 通信处理应用项目时，需要用到 Java 中的网络通信协议。在本章的内容中，将详细讲解在 Android 系统中开发 HTTP 通信项目的基本知识。

3.1　Java 中的网络通信基础

在互联网快速发展的今天，任何程序都难以离开网络，因为人们已经习惯了用网络快速传播信息。Java 提供了很多重要的协议来帮助人们开发网络项目，在 Android 系统开发网络应用时，需要用到 Java 中的这些协议。

3.1.1　Java网络通信概述

在学习 Java 网络通信之前，读者一定要弄清楚一些网络方面的专业术语，只有弄懂这些知识点，才能更好地掌握 Java 网络编程。

1. TCP/IP协议

TCP/IP是Transmission Control Protocol/Internet Protocol的简写，中文译名为传输控制协议/互联网络协议，是Internet最基本的协议，简单地说，就是由底层的IP协议和TCP协议组成的。众所周知，如今电脑上Internet都要作TCP/IP协议设置，显然该协议成了当今地球村“人与人”之间的“牵手协议”。

IP 协议的英文译名是互联网络协议，从这个名称就可以知道 IP 协议的重要性。在现实生活中，我们进行货物运输时都要把货物包装成一个个的纸箱或者是集装箱之后才进行运输，在网络世界中各种信息也是通过类似的方式进行传输的。IP 协议规定了数据传输时的基本单元和格式。如果比作货物运输，IP 协议规定了货物打包时的包装箱尺寸和包装的程序。除了这些以外，IP 协议还定义了数据包的递交办法和路由选择。同样用货物运输做比喻，IP 协议规定了货物的运输方法和运输路线。

从前面的介绍，读者可能已经知道了 IP 协议很重要，那 TCP 协议是做什么的呢？在 IP 协议中定义的传输是单向的，也就是说发出去的货物对方有没有收到我们是不知道的，就好像 8 毛钱一份的平信一样。那对于重要的信件我们要寄挂号信该怎么办呢？TCP 协议就是帮我们寄“挂号信”的。TCP 协议提供了可靠的面向对象的数据流传输服务的规则和约定。简单地说，在 TCP 模式中，对方发一个数据包给你，你要发一个确认数据包给对方，通过这种确认来提供可靠性。

2. 使用URL进行网络链接

URL 是 Uniform Resource Locator 的缩写，称为网页地址，是 Internet 上标准的资源的地址。它最初是由蒂姆·伯纳斯-李发明用来作为万维网的地址的。它已经被万维网联盟编制为因特网标准 RFC1738 了。它是用于完整地描述 Internet 上网页和其他资源的地址的一种标识方法。

Internet 上的每一个网页都具有一个唯一的名称标识，通常称之为 URL 地址，这种地址可以是本地磁盘，也可以是局域网上的某一台计算机，更多的是 Internet 上的站点。简单地说，URL 就是 Web 地址，俗称“网址”。

3.1.2　Socket和ServerSocket

在本节的学习中，用户将会学习网络编程的初步知识。网络编程可以分为创建Socket、打开连接 Socket 的输入流和输出流，对 Socket 进行编程，关闭 Socket。

1. 创建Socket

一个功能齐全的 Socket 的工作过程包含以下四个基本步骤。

(1) 创建 Socket。

(2) 打开连接到 Socket 的输入/输出流。

(3) 按照一定的协议对 Socket 进行读/写操作。

(4) 关闭 Socket。在实际应用中，并未使用到显示的 close，虽然很多文章都推荐如此，不过在简单的程序中，可能因为程序本身比较简单，所以无须使用 close。

在 Java.net 包中，定义了 Socket 和 ServerSocket 两个类，类 Socket 表示客户端或者服务器 Socket 连接的两端，类 Socket 中的构造方法如下。

- Socket(inetAddress address,int port)
- Socket(inetAddress address,int port,boolean stream)
- Socket(String host,int port)
- Socket(String host,int port,boolean stream)
- Socket(SocketImp imp1)
- Socket(String host, int port ,inetAddress localAddr,int localport)
- Socket(inetAddress address,int port, inetAddreess localAddr,int localport)

参数 address 表示 IP 地址，host 表示主机名，port 表示端口号，stream 用于指明 Socket 是流还是数据报，localPort 表示本地主机的端口号，localAddr 是本地及其地址，impl 是 Socket 的父类，在 SQLSocket 类中有以下三个构造方法。

- ServerSocket(int port)
- ServerSocket(int port,int backlog)
- ServerSocket(int port,int backlog,InetAddress bindAddr)

其中参数 bindAddr 表示本机地址。

Socket 和 ServerSocket 类库位于 java.net 包中。ServerSocket 用于服务器端，Socket 是建立网络连接时使用的。在连接成功时，应用程序两端都会产生一个 Socket 实例，操作这个实例，完成所需的会话。对于一个网络连接来说，套接字是平等的，并没有差别，不因为在服务器端或在客户端而产生不同级别。不管是 Socket 还是 ServerSocket，它们的工作都是通过 SocketImpl 类及其子类完成的。

java.net.Socket 继承于 java.lang.Object，有八个构造器，其方法并不多，下面介绍使用最频繁的三个方法，其他方法大家可以参考 JDK 文档。

- Accept 方法：用于产生“阻塞”，直到接收到一个连接，并且返回一个客户端的 Socket 对象实例。“阻塞”是一个术语，它使程序运行暂时“停留”在这个地

方，直到一个会话产生，然后程序继续执行；通常“阻塞”是由循环产生的。

- getInputStream 方法：获得网络连接输入，同时返回一个 IutputStream 对象实例。
- getOutputStream 方法：连接的另一端将得到输入，同时返回一个 OutputStream 对象实例。

注意：其中 getInputStream 和 getOutputStream 方法均会产生一个 IOException，它必须被捕获，因为它们返回的流对象通常都会被另一个流对象使用。

下面通过两段程序来讲解 Socket 的用法，一个是服务器端程序；另一个是客户端程序。

实　例	功　能	源码路径
实例 3-1	演示实现客户端和服务器端通信的过程	下载路径:\daima\3\KeSocket.java 下载路径:\daima\3\ServerSocket1.java

客户端程序 KeSocket.java 的实现代码如下：

```
import java.net.*;
import java.io.*;
public class KeSocket
{
    public static void main(String args[])
    {
        try
        {
            Socket s=new Socket("127.0.0.1",2007);
            InputStream is=s.getInputStream();
            OutputStream os=s.getOutputStream();
            PrintStream ps=new PrintStream(os);
            ps.println("hello");
            DataInputStream dis=new DataInputStream(is);
            String request=dis.readLine();
            System.out.println(request);
            s.close();
        }
        catch(ConnectException e)
        {
            System.out.println("异常"+e);
        }
        catch(IOException ee)
        {
            System.out.println("异常"+ee);
        }
        catch(Exception eee)
        {
            System.out.println("异常"+eee);
        }
    }
}
```

服务器端程序 ServerSocket1.java 的实现代码如下：

```
import java.net.*;
import java.io.*;
public class ServerSocket1
{
    public static void main(String args[])
    {
        try
        {
            ServerSocket ss=new ServerSocket(2007);
            while(true)
            {
                Socket s=ss.accept();
                InputStream is=s.getInputStream();
                OutputStream os=s.getOutputStream();
                DataInputStream dis=new DataInputStream(is);
                String request=dis.readLine();
                System.out.println(request);
                PrintStream ps=new PrintStream(os);
                ps.println("Bye");
                s.close();
                ss.close();
            }
        }
        catch(IOException e)
        {
            System.out.println("异常"+e);
        }
        catch(Exception ee)
        {
            System.out.println("异常"+ee);
        }
    }
}
```

先执行服务器端程序，执行效果如图 3-1 所示。



图 3-1　服务器端执行结果

执行客户端程序，执行效果如图 3-2 所示。

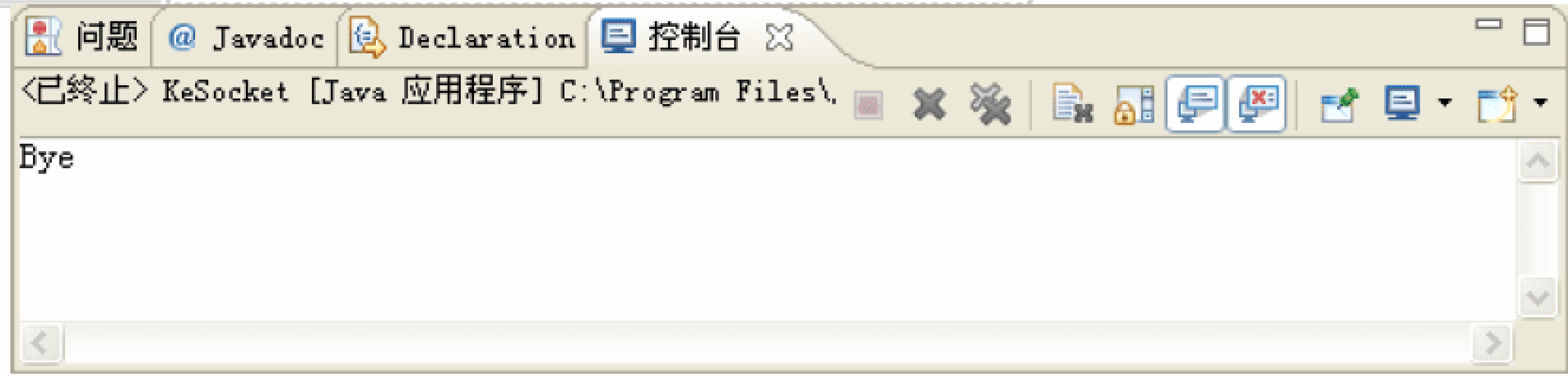


图 3-2　客户端执行的程序

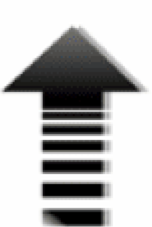
2. 多客户连接

在实际的应用中，往往有许多客户端向服务器发送请求，服务器端会启动一个专门的服务线程来响应客户端的请求，同时服务器本身在启动线程之后，马上进入监听状态。下面通过一段代码讲解多客户请求服务器的程序，服务器端程序如下。

实　例	功　能	源码路径
实例 3-2	演示多客户请求服务器的过程	下载路径:\daima\3\DuoServer.java

实例文件 DuoServer.java 的主要代码如下：

```
import java.net.*;
import java.io.*;
public class DuoServer
{
    public static void main(String args[])
    {
        boolean connected=true;
        try
        {
            ServerSocket ss=new ServerSocket(2007);
            int clientnum=0;
            while(connected)
            {
                ServerThread st=new ServerThread(ss.accept(),clientnum);
                st.start();
                clientnum++;
                System.out.println(clientnum);
            }
        }
        catch(Exception e)
        {
            System.out.println("异常"+e);
        }
    }
}
class ServerThread extends Thread
{
    private Socket s;
    int clientnum;
    public ServerThread(Socket s,int num)
    {
```



```
        this.s=s;
        clientnum=num+1;
    }
    public void run()
    {
        try
        {
            InputStream is=s.getInputStream();
            OutputStream os=s.getOutputStream();
            DataInputStream dis=new DataInputStream(is);
            String request=dis.readLine();
            System.out.println(request);
            PrintStream ps=new PrintStream(os);
            ps.println("Bye");
            s.close();
        }
        catch(Exception e)
        {
            System.out.println("异常"+e);
        }
    }
}
```

运行本实例程序，然后运行上一节的客户端程序的代码，调用 7 次后就会看到如图 3-3 所示的结果。

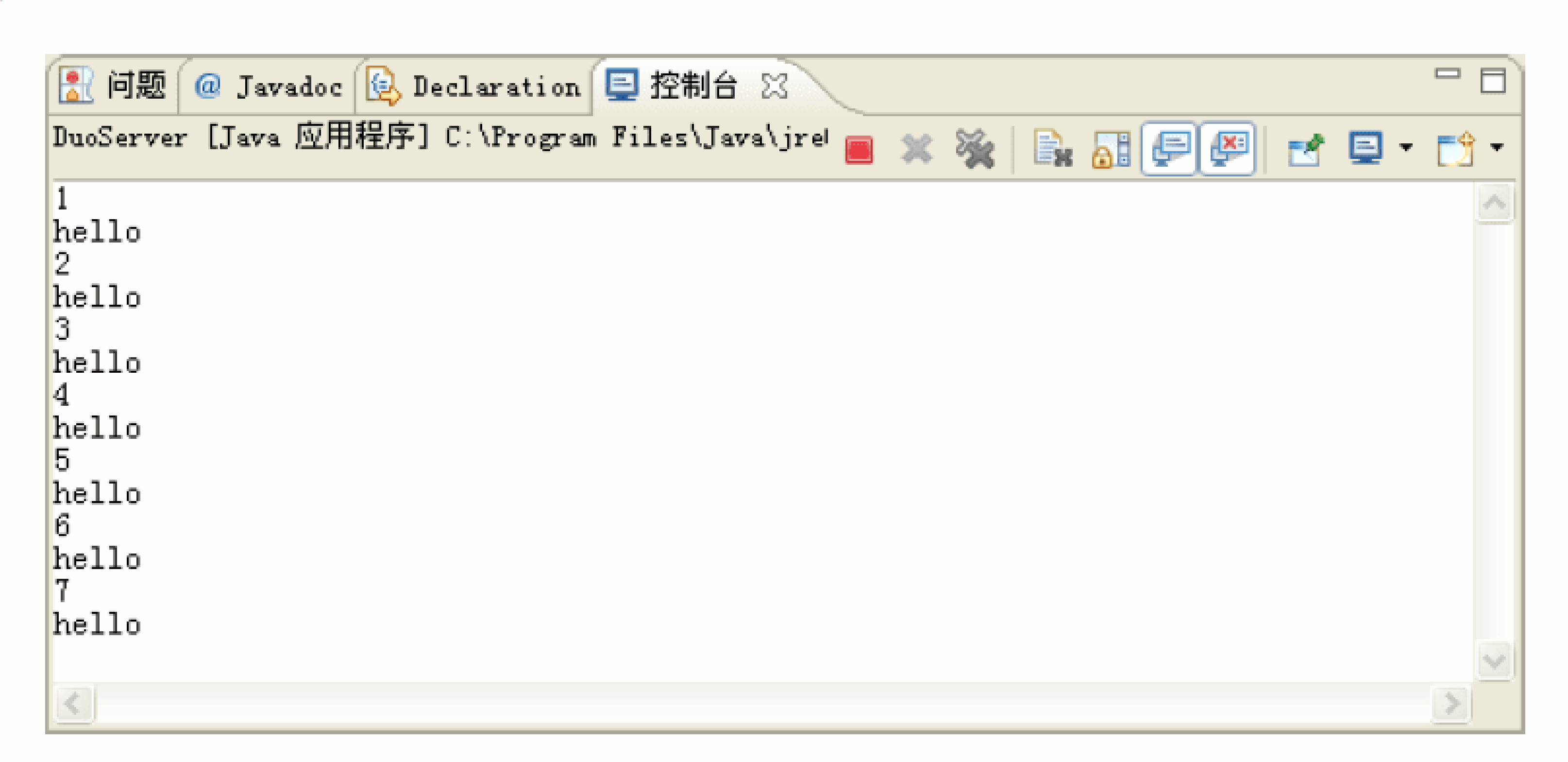


图 3-3　多客户连接

3.1.3　网络通信的综合应用

经过前面内容的学习，我们了解了 Java 技术中实现网络通信的基本知识。下面将通过一个具体实例的实现过程，讲解实现客户端和服务器端通信的流程。

实　例	功　能	源码路径
实例 3-3	演示客户端和服务器端的通信过程	下载路径:\daima\3\client.java 下载路径:\daima\3\Server.java

服务器端的实现文件是 Server.java，主要代码如下：




```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.util.*;

class Conn extends Thread{
    private JTextArea txt;
    private Socket st;
    private String msg = null;
    private BufferedReader br = null;
    private PrintStream ps;
    public Conn(Socket st,JTextArea txt){
        this.st = st;
        this.txt = txt;
        start();
    }
    public void run(){
        try{
         br = new BufferedReader(new InputStreamReader(new DataInputStream
              (st.getInputStream())));
        ps = new PrintStream(new DataOutputStream(st.getOutputStream()));
        }catch(Exception e){
            System.err.println("input failed");
        }
        while(true){
            try{
                msg = br.readLine();
                txt.append("从客户端收到信息"+msg+'\n');
                Server.send(msg);
            }catch(Exception e){
                System.err.println("connection closed");
                break;
            }
        }
    }

    public void send(String msg){
      ps.println(msg);
    }
}
public class Server extends JFrame{
    private JTextArea txt;
    private ServerSocket ss;
    private static java.util.List<Conn> conns = new ArrayList<Conn>();
    public Server(){
        txt=new JTextArea();
        this.setTitle("服务器");
        this.setLayout(new BorderLayout());
        this.add(new JScrollPane(txt),BorderLayout.CENTER);
```

```
        this.setSize(500,300);
        this.setVisible(true);
        run();
    }
    public void run(){
        try{
            ss = new ServerSocket(8000);
        }catch(Exception e){
            System.err.println("open socket failed");
        }
        txt.append("信息接收时间是:"+new Date()+"\n");
        while(true){
            try{
                Socket st=ss.accept();
                conns.add(new Conn(st,txt));
            }
            catch(IOException ex){
                System.err.println(ex);
            }
        }
}
    public static void send(String msg){
      for(Conn c:conns)
        c.send(msg); }
    public static void main(String args[]){
        Server myserver=new Server();
    }
}
```

客户端的实现文件是 client.java，主要代码如下：

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.util.*;
public class Client extends JFrame implements ActionListener
{
    final JTextArea txta;
    JTextField txtf;
    JPanel pl;
    JButton bt;
    BufferedReader br;
    DataOutputStream out;
    PrintStream ps;
    Container f=this.getContentPane();
        public Client(){
        f.setLayout(new BorderLayout());
        txta=new JTextArea();
        f.add(txta,BorderLayout.CENTER);
```

```
        txtf=new JTextField(20);
        bt=new JButton("发送");
        pl=new JPanel();
        pl.setLayout(new FlowLayout());
        pl.add(txtf);
        pl.add(bt);
        bt.addActionListener(this);
        f.add(pl,BorderLayout.SOUTH);
        setTitle("信息发送端");
        setSize(500,300);
        setVisible(true);
        run();
        new Thread(){
          {start();}
          public void run(){
            while(true){
              try{
                txta.append("收到消息: "+br.readLine()+"\n");
              }catch(Exception ex){}
            }
  }
 }
 }
   public void actionPerformed(ActionEvent e){
       if(e.getSource()==bt){
           String msg=txtf.getText();
           try{
               ps.println(msg);
               txta.append("已经发送消息:"+msg+"\n");
                          }
           catch(Exception ex){
               txta.append(ex.toString()+'\n');          }
       }
 }
   public void run(){
       try{
           Socket sc=new Socket("127.0.0.1",8000);
           out=new DataOutputStream(sc.getOutputStream());
           ps = new PrintStream(out);
 br = new BufferedReader(new InputStreamReader
(new DataInputStream(sc.getInputStream())));
       }
       catch(IOException ex){
           txta.append(ex.toString()+'\n');
       }}
   public static void main(String args[]) {
       Client myclient=new Client();
   }
}
```

在调试时，需要先执行客户端程序 Server.java，执行效果如图 3-4 所示。然后执行服

务器端程序 client.java，执行效果如图 3-5 所示。

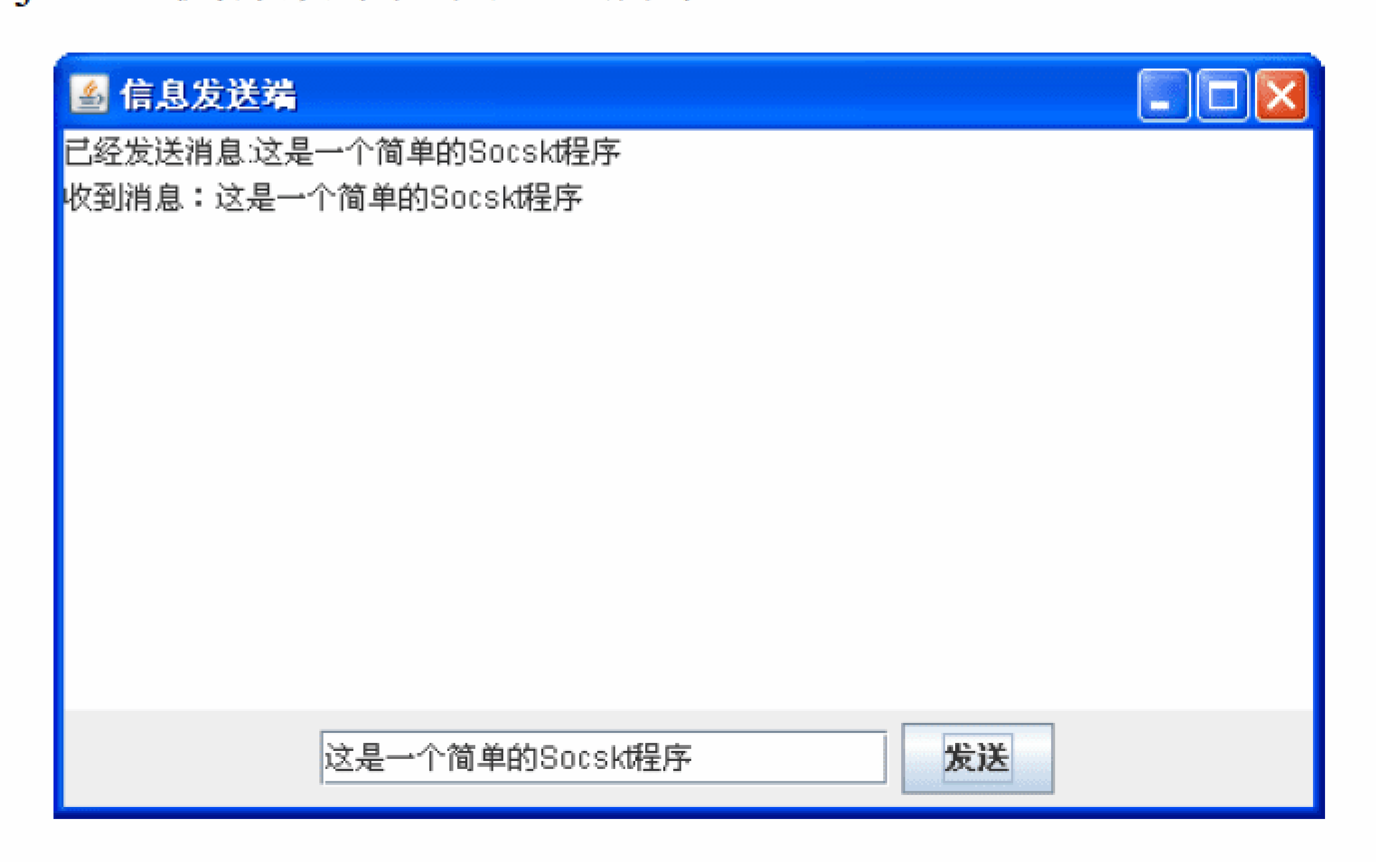


图 3-4　客户器端运行结果

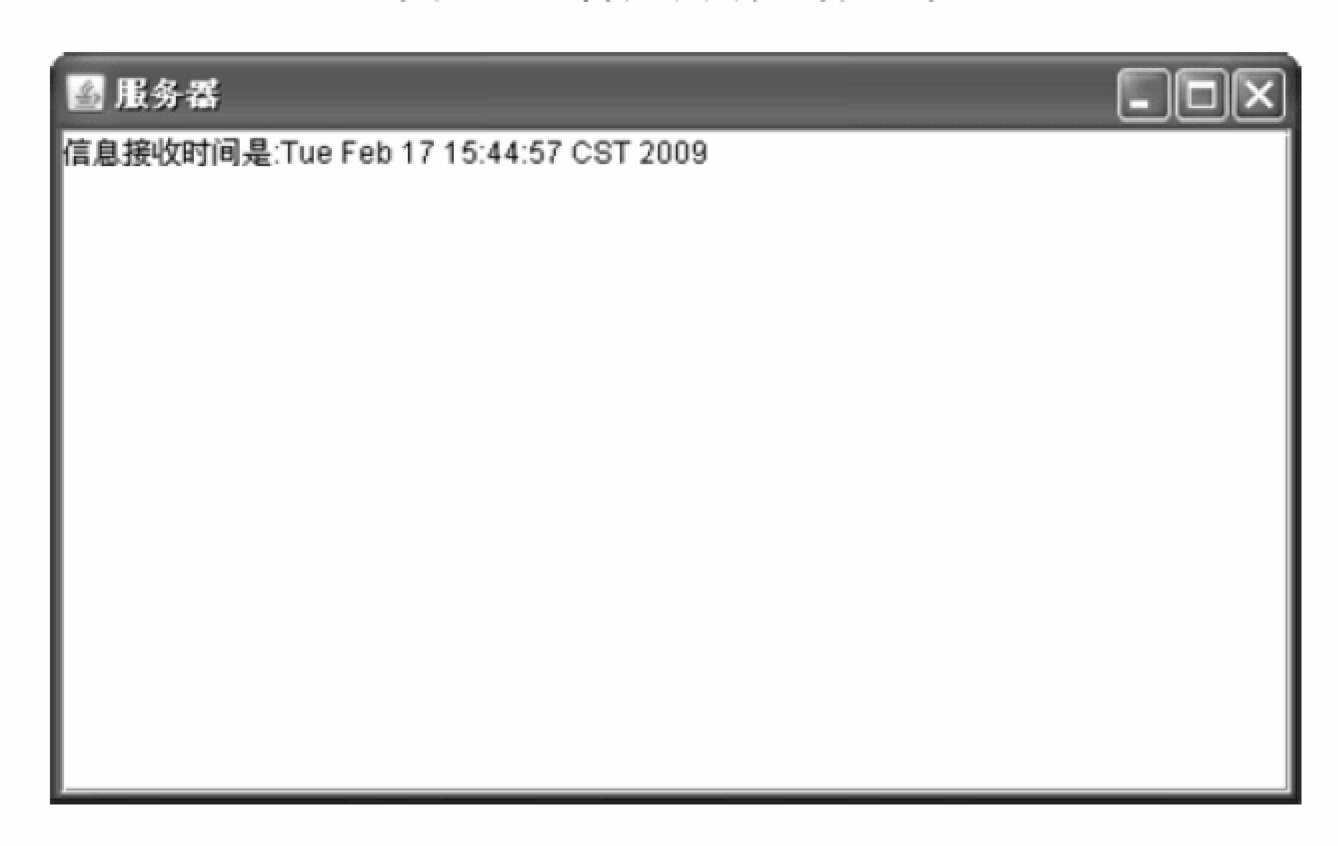


图 3-5　服务器端程序

3.2　HTTP 协议

HTTP 意为超文本传输协议，是 HyperText Transfer Protocol 的缩写，是互联网上应用最为广泛的一种网络协议，所有的 WWW 文件都必须遵守这个标准。设计 HTTP 的最初目的是为了提供一种发布和接收 HTML 页面的方法。在本节将简要介绍 HTTP 技术的相关基本理论知识，为读者步入本书后面知识的学习打下基础。

3.2.1　HTTP概述

HTTP 是一个客户端和服务器端请求和应答的标准(TCP)。客户端是终端用户，服务器端是网站。通过使用 Web 浏览器、网络爬虫或者其他工具，客户端发起一个到服务器上指定端口(默认端口为 80)的 HTTP 请求。我们称这个客户端为用户代理(User Agent)。应答的服务器上存储着(一些)资源，比如 HTML 文件和图像，我们称这个应答服务器为源服务器(Origin Server)。在用户代理和源服务器中间可能存在多个中间层，比如代理、网关或者隧

道(tunnels)。尽管 TCP/IP 协议是互联网上最流行的应用，HTTP 协议并没有规定必须使用它和(基于)它支持的层。事实上，HTTP 可以在任何其他互联网协议上，或者在其他网络上实现。HTTP 只假定(其下层协议提供)可靠的传输，任何能够提供这种保证的协议都可以被其使用。

通常，由 HTTP 客户端发起一个请求，建立一个到服务器指定端口(默认是 80 端口)的 TCP 连接。HTTP 服务器则在那个端口监听客户端发送过来的请求。一旦收到请求，服务器(向客户端)发回一个状态行，比如“HTTP/1.1 200 OK”，和响应的消息，消息的消息体可能是请求的文件、错误消息或者其他一些信息。

HTTP 使用 TCP 而不是 UDP 的原因在于(打开)一个网页必须传送很多数据，而 TCP 协议提供传输控制，按顺序组织数据和进行错误纠正。

3.2.2 协议功能

HTTP 是超文本传输协议，是客户端浏览器或其他程序与 Web 服务器之间的应用层通信协议。在 Internet 上的 Web 服务器上存放的都是超文本信息，客户机需要通过 HTTP 协议传输所要访问的超文本信息。HTTP 包含命令和传输信息，不仅可用于 Web 访问，也可以用于其他因特网/内联网应用系统之间的通信，从而实现各类应用资源超媒体访问的集成。

当我们想浏览一个网站的时候，只需在浏览器的地址栏里输入网站的地址就可以了，例如，www.*****.com，但是在浏览器的地址栏里面出现的却是：http://www.*******，读者知道为什么会多出一个“http”吗？

我们在浏览器的地址栏里输入的网站地址叫作 URL(Uniform Resource Locator，统一资源定位符)。就像每家每户都有一个门牌地址一样，每个网页也都有一个 Internet 地址。当在浏览器的地址栏中输入一个 URL 或是单击一个超级链接时，URL 就确定了要浏览的地址。浏览器通过超文本传输协议(HTTP)将 Web 服务器上站点的网页代码提取出来，并翻译成漂亮的网页。因此，在我们认识 HTTP 之前，有必要先弄清楚 URL 的组成。例如，http://www.****.com /china/index.htm 的含义如下。

- http://：代表超文本转移协议，通知****.com 服务器显示 Web 页，通常不用输入。
- www：代表一个 Web(万维网)服务器。
- ****.com/：这是装有网页的服务器的域名或站点服务器的名称。
- china/：为该服务器上的子目录，就好像我们的文件夹。
- index.htm：是文件夹中的一个 HTML 文件(网页)。

众所周知，Internet 的基本协议是 TCP/IP 协议，然而在 TCP/IP 模型最上层的是应用层(Application layer)，它包含所有高层的协议。高层协议有：文件传输协议 FTP、电子邮件传输协议 SMTP、域名系统服务 DNS、网络新闻传输协议 NNTP 和 HTTP 协议等。

HTTP 协议(HyperText Transfer Protocol，超文本传输协议)是用于从 WWW 服务器传输超文本到本地浏览器的传输协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。这就是为什么在浏览器中看到的网页地址都是以“http://”开头的原因。

3.2.3　Android中的HTTP

在 Android 系统中，提供了以下三种通信接口。

- 标准 Java 接口：java.net。
- Apache 接口：org.apache.http。
- Android 网络接口：android.net.http。

网络编程在无线应用程序开发过程中起到了重要的作用。在 Android 系统中包括 Apache HttpClient 库，此库为执行 Android 中的网络操作之首选方法。除此之外，Android 还可允许通过标准的 Java 联网 API(java.net 包)来访问网络。即便使用 Java.net 包，也是在内部使用该 Apache 库。

为了访问互联网，需要设置应用程序获取 android.permission.INTERNET 权限的许可在 Android 系统中，存在如下与网络连接相关的包。

1) java.net

该包提供联网相关的类，包括流和数据报套接字、互联网协议以及通用的 HTTP 处理。此为多用途的联网资源。经验丰富的 Java 开发人员可立即使用此惯用的包来创建应用程序。

2) java.io

尽管未明确联网，但其仍然非常重要。此包中的各种类通过其他 Java 包中提供的套接字和链接来使用。它们也可用来与本地文件进行交互(与网络进行交互时经常发生)。

3) java.nio

该包包含表示具体数据类型的缓冲的各种类。便于基于 Java 语言的两个端点之间的网络通信。

4) org.apache.*

表示可为进行 HTTP 通信提供精细控制和功能的各种包。可将 Apache 识别为普通的开源 Web 服务器。

5) android.net

该包包括核心 java.net.*类之外的各种附加的网络接入套接字，包括 URL 类，其通常在传统联网之外的 Android 应用程序开发中使用。

6) android.net.http

该包包含可操作 SSL 证书的各种类。

7) android.net.wifi

该包包含可管理 Android 平台中 Wi-Fi(802.11 无线以太网)所有方面的各种类。并非所有的设备均配备有 Wi-Fi 能力，尤其随着 Android 在对制造商(如诺基亚和 LG)手机的翻盖手机研发方面取得了进展。

8) android.telephony.gsm

该包包含管理和发送短信(文本)消息所要求的各种类。随着时间的推移，可能将引入一种附加的包，以提供有关非 GSM 网络(如 CDMA 或类似 android.telephony.cdma)的类似功能。

3.3 使用 Apache 接口

Apache HttpClient 是一个开源项目，弥补了 java.net.* 灵活性不足的缺点，为客户端的 HTTP 编程提供高效、最新、功能丰富的工具包支持。Android 平台引入了 Apache HttpClient 的同时还提供了对它的一些封装和扩展，例如设置默认的 HTTP 超时和缓存大小等。早期的 Android 还同时包括 Commons HttpClient(org.apache.commons.httpclient.*)和 HttpComponents (org.apache.http.client.*)，因为在 Android 平台中，使用得最多的是 Apache 接口。本节将详细介绍使用 Apache 接口(org.apache.http)实现和网络连接的基本知识，希望读者结合演示代码深入理解每一个知识点，做到学以致用。

3.3.1 Apache接口基础

在 Apache HttpClient 库中，以下内容是对网络连接有用的各种包。

- org.apache.http.HttpResponse
- org.apache.http.client.HttpClient
- org.apache.http.client.methods.HttpGet
- org.apache.http.impl.client.DefaultHttpClient
- HttpClient httpclient=new DefaultHttpClient();

如果想从服务器检索此信息，则需要使用 HttpGet 类的构造器，例如下面的代码：

```
HttpGet request=new HttpGet("http://innovator.samsungmobile.com");
```

然后用 HttpClient 类的 execute()方法中的 HttpGet 对象来检索 HttpResponse 对象，例如下面的代码：

```
HttpResponse response = client.execute(request);
```

接着读取已检索的响应，例如下面的代码：

```
BufferedReader rd = new BufferedReader
                              (new
InputStreamReader(response.getEntity().getContent()));
   String line = "";
   while ((line = rd.readLine()) != null) {
      Log.d("output: ",line);
   }
```

3.3.2 Apache应用基础

1. 连网流程

在 Android 系统中，可以采用 HttpPost 和 HttpGet 来封装 Post 请求和 Get 请求，然后再使用 HttpClient 的 excute()方法发送 Post 或者 Get 请求并返回服务器的响应数据。使用



Apache 连网的基本流程如下。

(1) 设置连接和读取超时时间，并新建 HttpClient 对象，例如下面的代码：

```
// 设置连接超时时间和数据读取超时时间
HttpParams httpParams = new BasicHttpParams();
HttpConnectionParams.setConnectionTimeout(httpParams,
        KeySource.CONNECTION TIMEOUT INT);
HttpConnectionParams.setSoTimeout(httpParams,
        KeySource.SO_TIMEOUT_INT);
//新建 HttpClient 对象
HttpClient httpClient = new DefaultHttpClient(httpParams)
```

(2) 实现 Get 请求，例如下面的代码：

```
// 获取请求
 HttpGet get = new HttpGet(url);
 // set HTTP head parameters
 //Map<String, String> headers
 if (headers != null)
 {
     Set<String> setHead = headers.keySet();
     Iterator<String> iteratorHead = setHead.iterator();
     while (iteratorHead.hasNext())
     {
         String headerName = iteratorHead.next();
         String headerValue = (String) headers.get(headerName);
         MyLog.d(headerName, headerValue);
         get.setHeader(headerName, headerValue);
     }
 }
     // connect
     //need try catch
     response = httpClient.execute(get);
```

(3) 实现 Post 发送请求处理，例如下面的代码：

```
    HttpPost post = new HttpPost(KeySource.HOST URL STR);
     // set HTTP head parameters
     Map<String, String> headers = heads;
     Set<String> setHead = headers.keySet();
     Iterator<String> iteratorHead = setHead.iterator();
     while (iteratorHead.hasNext())
     {
         String headName = iteratorHead.next();
         String headValue = (String) headers.get(headName);
         post.setHeader(headName, headValue);
     }
   /**
          * 通常的 HTTP 实体需要在执行上下文的时候动态生成的。
          * HttpClient 的提供使用 EntityTemplate 实体类和 ContentProducer
          接口支持动态实体。
```

```
         * 内容制作是通过写需求的内容到一个输出流，每次请求的时候都会产生。
         * 因此，通过 EntityTemplate 创建实体通常是独立的，重复性好。
         */
    ContentProducer cp = new ContentProducer()
        {
            public void writeTo(OutputStream outstream)
                    throws IOException
            {
                Writer writer = new OutputStreamWriter(outstream,
                        "UTF-8");
                writer.write(requestBody);
                writer.flush();
                writer.close();
            }
        };
        HttpEntity entity = new EntityTemplate(cp);
        post.setEntity(entity);
    }
    //connect , need try catch
    response = httpClient.execute(post);
```

(4) 通过 Response 响应请求，例如下面的代码：

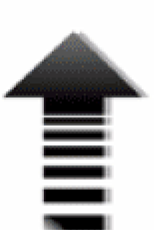
```
if (response.getStatusLine().getStatusCode() == 200)
          {
              /**
          * 因为直接调用 toString 可能会导致某些中文字符出现乱码的情况。所以此处使用
            toByteArray
          * 如果需要转成 String 对象，可以先调用 EntityUtils.toByteArray()方法将
            消息实体转成 byte 数组，
          * 再由 new String(byte[] bArray)转换成字符串。
               */
              byte[] bResultXml = EntityUtils.toByteArray(response
                      .getEntity());
              if (bResultXml != null)
              {
                      String  strXml = new String(bResultXml, "utf-8");
              }
         }
```

这样，使用 Apache 实现连网处理数据交互的过程就完成了。无论多么复杂的项目，都必须遵循上面的流程。

2. HttpClient网络通信

Apache 的核心功能是 HttpClient，与网络有关的功能几乎都需要用 HttpClient 实现。因为在 Android 开发过程中，经常会用到网络连接功能与服务器进行数据的交互，所以 Android 的 SDK 提供了 Apache 的 HttpClient 来方便我们使用各种 Http 服务。我们可以把 HttpClient 想象成一个浏览器，通过它的 API，可以很方便地发出 Get 请求和 Post 请求。

例如只需要以下几行代码就能发出一个简单的 Get 请求并打印响应结果。



```
try {
        // 创建一个默认的 HttpClient
        HttpClient httpclient = new DefaultHttpClient();
        // 创建一个 Get 请求
        HttpGet request = new HttpGet("www.google.com");
        // 发送 Get 请求，并将响应内容转换成字符串
        String response = httpclient.execute(request, new
           BasicResponseHandler());
        Log.v("response text", response);
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

有读者可能会问为什么上述代码要使用单例 HttpClient 呢？这只是一段演示代码，实际项目中的请求与响应处理会复杂一些，并且还要考虑到代码的容错性，但这并不是本篇的重点。读者重点注意代码的第三行：

```
HttpClient httpclient = new DefaultHttpClient();
```

在发出 HTTP 请求前先创建了一个 HttpClient 对象，而在实际项目中，我们很可能会在多处进行 HTTP 通信，这时候不需要为每个请求都创建一个新的 HttpClient。因为之前已经提到，HttpClient 就像一个小型的浏览器，对于整个应用，只需要一个 HttpClient 就够了。由此可以得出，使用简单的单例就可以实现，例如下面的代码：

```
public class CustomerHttpClient {
    private static HttpClient customerHttpClient;
    private CustomerHttpClient() {
    }

    public static HttpClient getHttpClient() {
        if(null == customerHttpClient) {
            customerHttpClient = new DefaultHttpClient();
        }
        return customerHttpClient;
    }
}
```

但是如果同时有多个请求需要处理呢？答案是使用多线程。假如现在应用程序使用同一个 HttpClient 来管理所有的 Http 请求，一旦出现并发请求，那么一定会出现多线程的问题。这就好像我们的浏览器只有一个标签页却有多个用户，A 要上 Google，B 要上 baidu，这时浏览器就会忙不过来。幸运的是，HttpClient 提供了创建线程安全对象的 API，帮助我们很快地得到线程安全的“浏览器”。例如下面的代码很好地解决了多线程问题：

```
public class CustomerHttpClient {
    private static final String CHARSET = HTTP.UTF_8;
    private static HttpClient customerHttpClient;
    private CustomerHttpClient() {
```

```
    }
    public static synchronized HttpClient getHttpClient() {
       if (null == customerHttpClient) {
          HttpParams params = new BasicHttpParams();
          // 设置一些基本参数
          HttpProtocolParams.setVersion(params, HttpVersion.HTTP 1 1);
          HttpProtocolParams.setContentCharset(params,
                CHARSET);
          HttpProtocolParams.setUseExpectContinue(params, true);
          HttpProtocolParams.setUserAgent(params,
             "Mozilla/5.0(Linux;U;Android 3.0;en-us;Nexus One Build.FRG83) "
                 + "AppleWebKit/553.1(KHTML,like Gecko) Version/4.0
                   Mobile Safari/533.1");
          // 超时设置
          /* 从连接池中取连接的超时时间 */
          ConnManagerParams.setTimeout(params, 1000);
          /* 连接超时 */
          HttpConnectionParams.setConnectionTimeout(params, 2000);
          /* 请求超时 */
          HttpConnectionParams.setSoTimeout(params, 4000);
          // 设置我们的 HttpClient 支持 HTTP 和 HTTPS 两种模式
          SchemeRegistry schReg = new SchemeRegistry();
          schReg.register(new Scheme("http", PlainSocketFactory
                .getSocketFactory(), 80));
          schReg.register(new Scheme("https", SSLSocketFactory
                .getSocketFactory(), 443));
          // 使用线程安全的连接管理来创建 HttpClient
          ClientConnectionManager conMgr = new
            ThreadSafeClientConnManager(
            params, schReg);
          customerHttpClient = new DefaultHttpClient(conMgr, params);
       }
       return customerHttpClient;
    }
}
```

在上面的代码中，通过 getHttpClient()方法为 HttpClient 配置了一些基本参数和超时设置，然后使用 ThreadSafeClientConnManager 来创建线程安全的 HttpClient。

3. HttpClient网络编程

下面来介绍使用 Apache HttpClient 库进行网络连接的过程，将通过一段具体代码的实现过程来讲解。

(1) 在文件 AndroidManifest.xml 中添加 android.permission.INTERNET 权限，这样才可以允许应用程序访问网络。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.apache"
     android:versionCode="1"
     android:versionName="1.0">
```

```
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.INTERNET"></uses-
       permission>
    <application android:icon="@drawable/icon"
       android:label="@string/app_name">
    <activity android:name=".ApacheConnection"
       android:label="@string/app name">
          <intent-filter>
             <action android:name="android.intent.action.MAIN" />
             <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
       </activity>
    </application>
</manifest>
```

(2) 使用 activity “ApacheConnection” 来创建项目 com.apache。将布局文件 main.xml 更改为如下代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
    <TextView
       android:text="Enter URL"
       android:id="@+id/textView1"
       android:layout width="wrap content"
       android:layout height="wrap content">
    </TextView>
    <EditText
       android:id="@+id/editText1"
       android:layout width="match parent"
       android:text="http://innovator.samsungmobile.com"
       android:layout height="wrap content">
    </EditText>
    <Button
       android:text="Click Here"
       android:id="@+id/button1"
       android:layout width="wrap content"
       android:layout height="wrap content">
     </Button>
    <EditText
       android:id="@+id/editText2"
       android:layout width="match parent"
       android:layout height="fill parent">
     </EditText>
</LinearLayout>
```

(3) 编写主程序文件 ApacheConnection.java，此代码将能允许查看 HTML 代码，具体实现代码如下：

```
package com.apache;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class ApacheConnection extends Activity {
Button bt;
 TextView textView1;
 TextView textView2;
/** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  bt = (Button) findViewById(R.id.button1);
  textView1 = (TextView) findViewById(R.id.editText1);
  textView2 = (TextView) findViewById(R.id.editText2);
bt.setOnClickListener(new OnClickListener() {
  @Override
  public void onClick(View v) {
      // TODO Auto-generated method stub
      textView2.setText("");
     try {
          /*Apache HttpClient Library*/
      HttpClient client = new DefaultHttpClient();
      HttpGet request = new HttpGet(textView1.getText().toString());
       HttpResponse response = client.execute(request);
      /* response code*/
      BufferedReader rd = new BufferedReader(
      new InputStreamReader(response.getEntity().getContent()));
     String line = "";
      while ((line = rd.readLine()) != null) {
            textView2.append(line);
     }
    } catch (Exception exe) {
        exe.printStackTrace();
                     }
          }
     });
 }
}
```

执行上述代码后，可以在手机浏览器中查看输入网址的网页 HTML 代码，如图 3-6

所示。

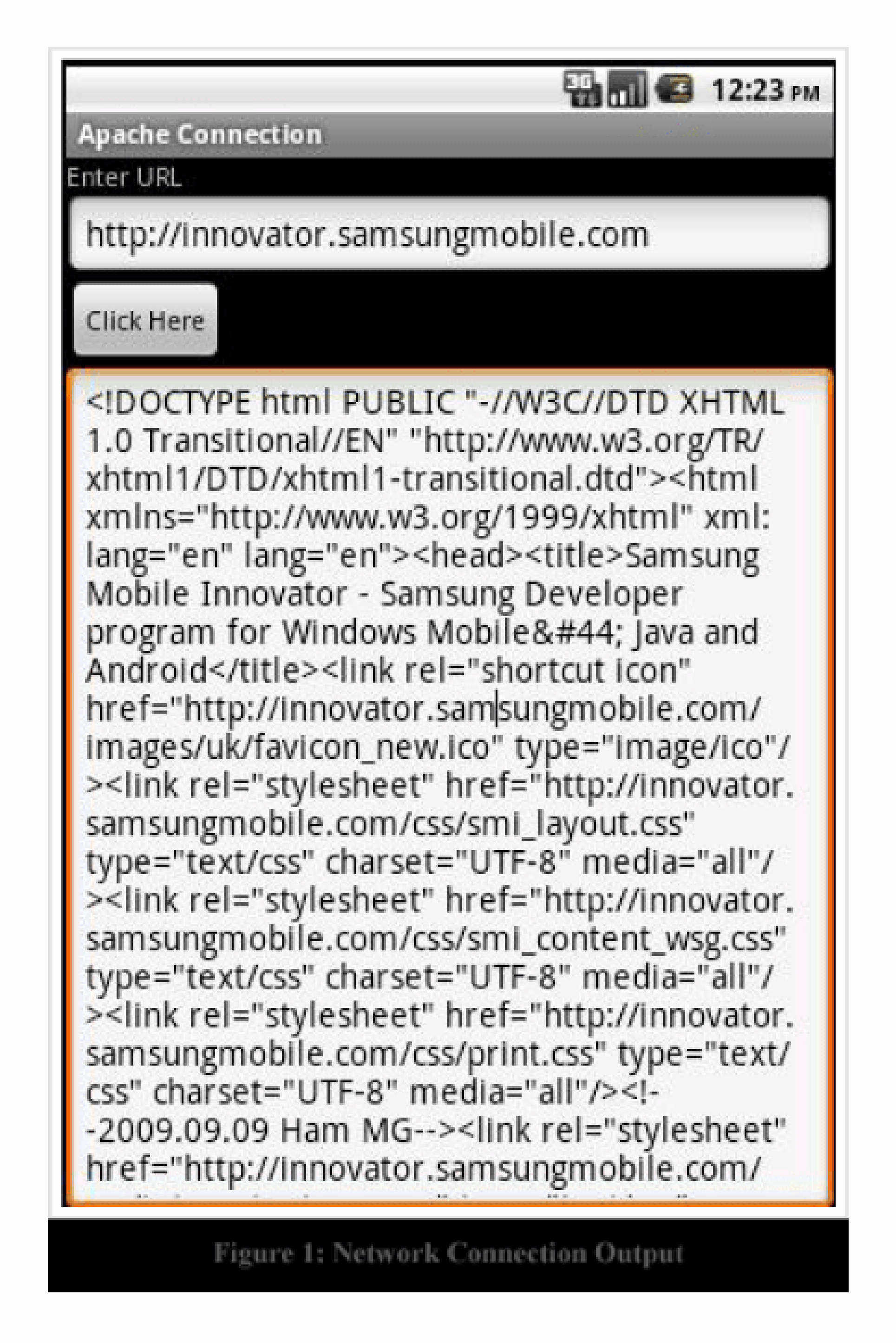


图 3-6　执行效果

3.3.3　Apache应用要点

Apache 中的 HttpClient 是一个完善的 HTTP 客户端，它提供了对 HTTP 协议的全面支持，可以使用 HTTP Get 方式和 Post 方式进行访问。下面结合实例，详细介绍使用 HttpClient 的方法。

(1) 新建一个 http 项目，项目结构如图 3-7 所示。

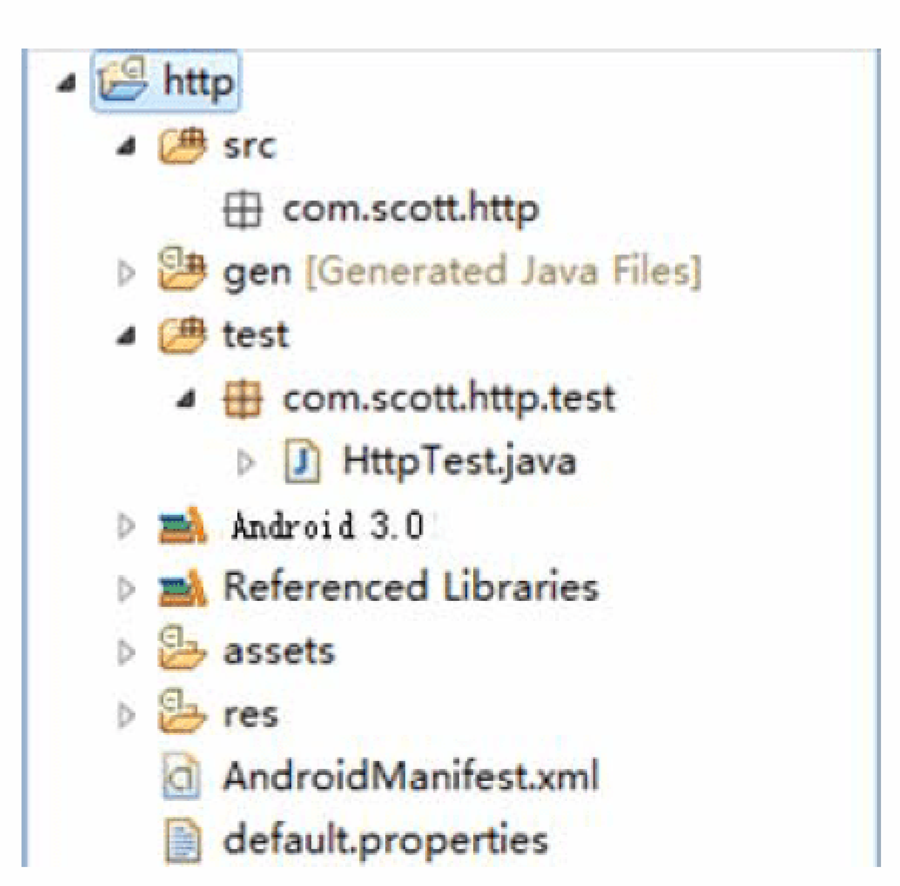


图 3-7　http项目结构

在这个项目中不需要任何的 Activity，所有的操作都在单元测试类 HttpTest.java 中完成。

(2) 因为使用到了单元测试，所以在这里先介绍如何配置 Android 中的单元测试。所有配置信息均在 AndroidManifest.xml 中完成，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.scott.http"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon"
        android:label="@string/app name">
       <!-- 配置测试要使用的类库 -->
       <uses-library android:name="android.test.runner"/>
   </application>
   <!-- 配置测试设备的主类和目标包 -->
   <instrumentation
android:name="android.test.InstrumentationTestRunner"
                    android:targetPackage="com.scott.http"/>
    <!-- 访问 HTTP 服务所需的网络权限 -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-sdk android:minSdkVersion="11" />
</manifest>
```

单元测试类需要继承于 android.test.AndroidTestCase 类，此类继承于 junit.framework.TestCase，并提供了 getContext()方法来获取 Android 上下文环境。

(3) 编写测试文件 HttpTest.java，具体代码如下：

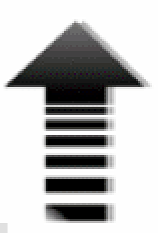
```
package com.scot.http.test;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

import junit.framework.Assert;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.mime.MultipartEntity;
import org.apache.http.entity.mime.content.InputStreamBody;
import org.apache.http.entity.mime.content.StringBody;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

import android.test.AndroidTestCase;
```



```
public class HttpTest extends AndroidTestCase {

    private static final String PATH = "http://192.168.1.57:8080/web";

    public void testGet() throws Exception {
        HttpClient client = new DefaultHttpClient();
        HttpGet get = new HttpGet(PATH +
            "/TestServlet?id=1001&name=john&age=60");
        HttpResponse response = client.execute(get);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC OK) {
            InputStream is = response.getEntity().getContent();
        String result = inStream2String(is);
        Assert.assertEquals(result, "GET SUCCESS");
    }
    }

    public void testPost() throws Exception {
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(PATH + "/TestServlet");
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("id", "1001"));
        params.add(new BasicNameValuePair("name", "john"));
        params.add(new BasicNameValuePair("age", "60"));
        HttpEntity formEntity = new UrlEncodedFormEntity(params);
        post.setEntity(formEntity);
        HttpResponse response = client.execute(post);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC OK) {
            InputStream is = response.getEntity().getContent();
        String result = inStream2String(is);
        Assert.assertEquals(result, "POST SUCCESS");
    }
    }

    public void testUpload() throws Exception {
        InputStream is = getContext().getAssets().open("books.xml");
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(PATH + "/UploadServlet");
        InputStreamBody isb = new InputStreamBody(is, "books.xml");
        MultipartEntity multipartEntity = new MultipartEntity();
        multipartEntity.addPart("file", isb);
        multipartEntity.addPart("desc", new StringBody("this is description."));
        post.setEntity(multipartEntity);
        HttpResponse response = client.execute(post);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC OK) {
            is = response.getEntity().getContent();
        String result = inStream2String(is);
        Assert.assertEquals(result, "UPLOAD SUCCESS");
    }
    }

    //将输入流转换成字符串
```

```
    private String inStream2String(InputStream is) throws Exception {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buf = new byte[1024];
        int len = -1;
        while ((len = is.read(buf)) != -1) {
            baos.write(buf, 0, len);
        }
        return new String(baos.toByteArray());
    }
}
```

在上述代码中包含了三个测试用例。

① 在定位服务器地址时使用到了 IP。因为服务端是在 Windows 上运行，这里不能用 localhost，而本单元测试运行在 Android 平台，如果使用 localhost 就意味着在 Android 内部去访问服务，可能会访问不到，所以必须用 IP 来定位服务。

② testGet 测试。使用 HttpGet 将请求参数直接附在 URL 后面，然后由 HttpClient 执行 Get 请求。如果响应成功则取得响应内如输入流，并转换成字符串，最后判断是否为 GET_SUCCESS。testGet 测试对应的服务端 Servlet 代码如下：

```
@Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        System.out.println("doGet method is called.");
        String id = request.getParameter("id");
        String name = request.getParameter("name");
        String age = request.getParameter("age");
        System.out.println("id:" + id + ", name:" + name + ", age:" + age);
        response.getWriter().write("GET SUCCESS");
    }
```

③ testPost 测试。在此使用 HttpPost，URL 后面并没有附带参数信息，参数信息被包装成一个由 NameValuePair 类型组成的集合形式，然后经过 UrlEncodedFormEntity 处理后调用 HttpPost 的 setEntity 方法进行参数设置，最后由 HttpClient 执行。testPost 测试对应的服务端代码如下：

```
@Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        System.out.println("doPost method is called.");
        String id = request.getParameter("id");
        String name = request.getParameter("name");
        String age = request.getParameter("age");
        System.out.println("id:" + id + ", name:" + name + ", age:" + age);
        response.getWriter().write("POST SUCCESS");
    }
```

上面的两段代码是最基本的 Get 请求和 Post 请求，参数都是文本数据类型，能满足普通的需求，不过在有的场合(例如要用到上传文件的时候)却不能使用基本的 Get 请求和 Post 请求了，我们要使用多部件的 Post 请求。下面介绍一下如何使用多部件 Post 操作上



传一个文件到服务端。

因为 Android 附带的 HttpClient 版本暂不支持多部件 Post 请求，所以需要用到一个 HttpMime 开源项目，该组件是专门处理与 MIME 类型有关的操作。因为 HttpMime 是包含在 HttpComponents 项目中的，所以我们需要去 Apache 官方网站下载 HttpComponents，然后把其中的 HttpMime.jar 包放到项目中去，如图 3-8 所示。

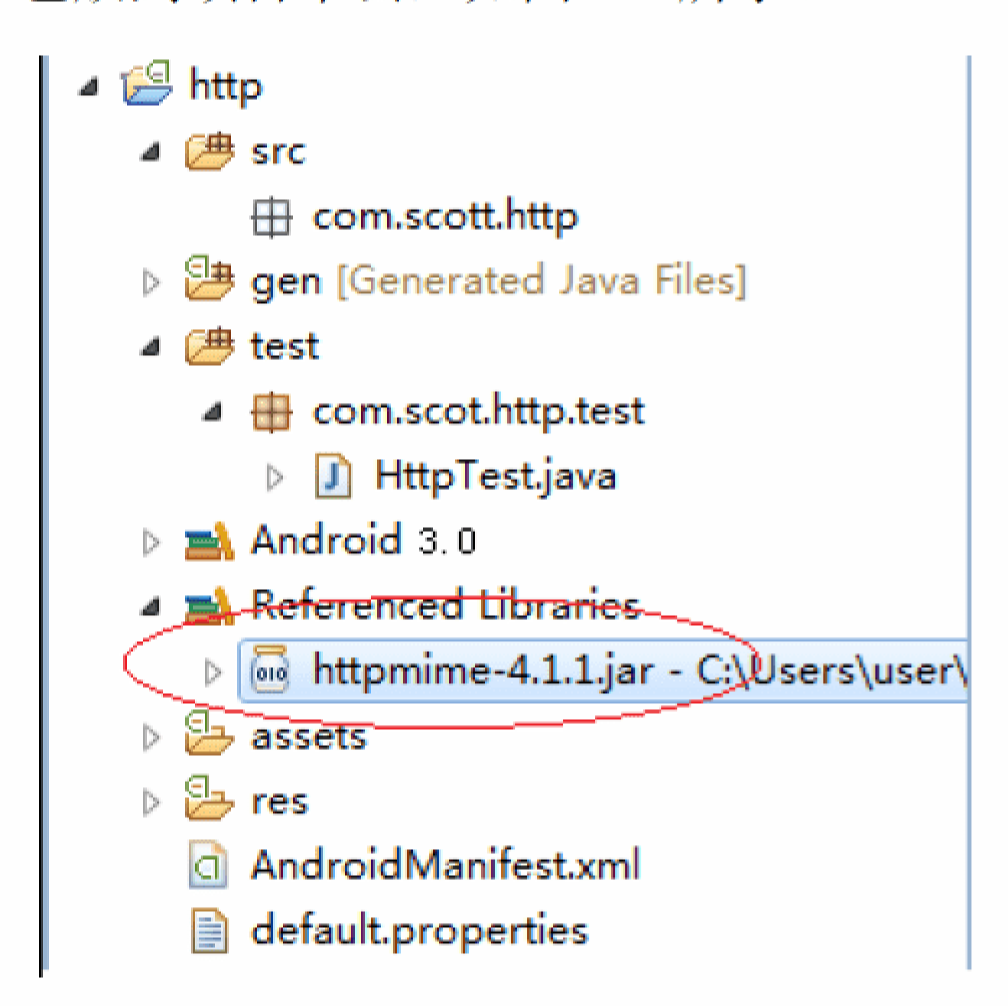


图 3-8　添加HttpMime.jar包

接下来看一下 testUpload 中的测试用例。首先用 HttpMime 提供的 InputStreamBody 处理文件流参数，然后用 StringBody 处理普通文本参数，最后把所有类型参数都加入到一个 MultipartEntity 的实例中，并将这个 MultipartEntity 设置为此次 Post 请求的参数实体，然后执行 Post 请求。服务端 Servlet 代码如下：

```
package com.scott.web.servlet;

import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

@SuppressWarnings("serial")
public class UploadServlet extends HttpServlet {
```

```
    @Override
    @SuppressWarnings("rawtypes")
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        boolean isMultipart = ServletFileUpload.isMultipartContent(request);
        if (isMultipart) {
            FileItemFactory factory = new DiskFileItemFactory();
            ServletFileUpload upload = new ServletFileUpload(factory);
            try {
                List items = upload.parseRequest(request);
                Iterator iter = items.iterator();
                while (iter.hasNext()) {
                    FileItem item = (FileItem) iter.next();
                    if (item.isFormField()) {
                     //普通文本信息处理
                     String paramName = item.getFieldName();
                        String paramValue = item.getString();
                        System.out.println(paramName + ":" + paramValue);
                    } else {
                     //上传文件信息处理
                        String fileName = item.getName();
                        byte[] data = item.get();
                        String filePath = getServletContext().getRealPath
                           ("/files") + "/" + fileName;
                        FileOutputStream fos = new FileOutputStream(filePath);
                        fos.write(data);
                        fos.close();
                    }
                }
            } catch (FileUploadException e) {
                e.printStackTrace();
            }
        }
        response.getWriter().write("UPLOAD SUCCESS");
    }
}
```

这样，服务端就成功使用 Apache 开源项目 FileUpload 实现了文件上传处理。在使用时一定不要忘记附加 commons-fileupload 和 commons-io 这两个项目的 jar 包，对服务端开发不太熟悉的朋友可以到网上查找一下相关资料。

介绍完上面的三种不同的情况之后还需要考虑一个问题，在实际项目中我们不可能每次都新建 HttpClient，而是应该只为整个应用创建一个 HttpClient，这样就可以将其用于所有 HTTP 通信。另外还需要注意，在通过一个 HttpClient 同时发出多个请求时可能会引发多线程问题。针对上述两个问题，需要优化处理上述项目，优化处理过程如下。

① 扩展系统默认的 Application，并将其应用在项目中。

② 使用 HttpClient 类库提供的 ThreadSafeClientManager 来创建和管理 HttpClient。优化处理后的工程文件结构如图 3-9 所示。



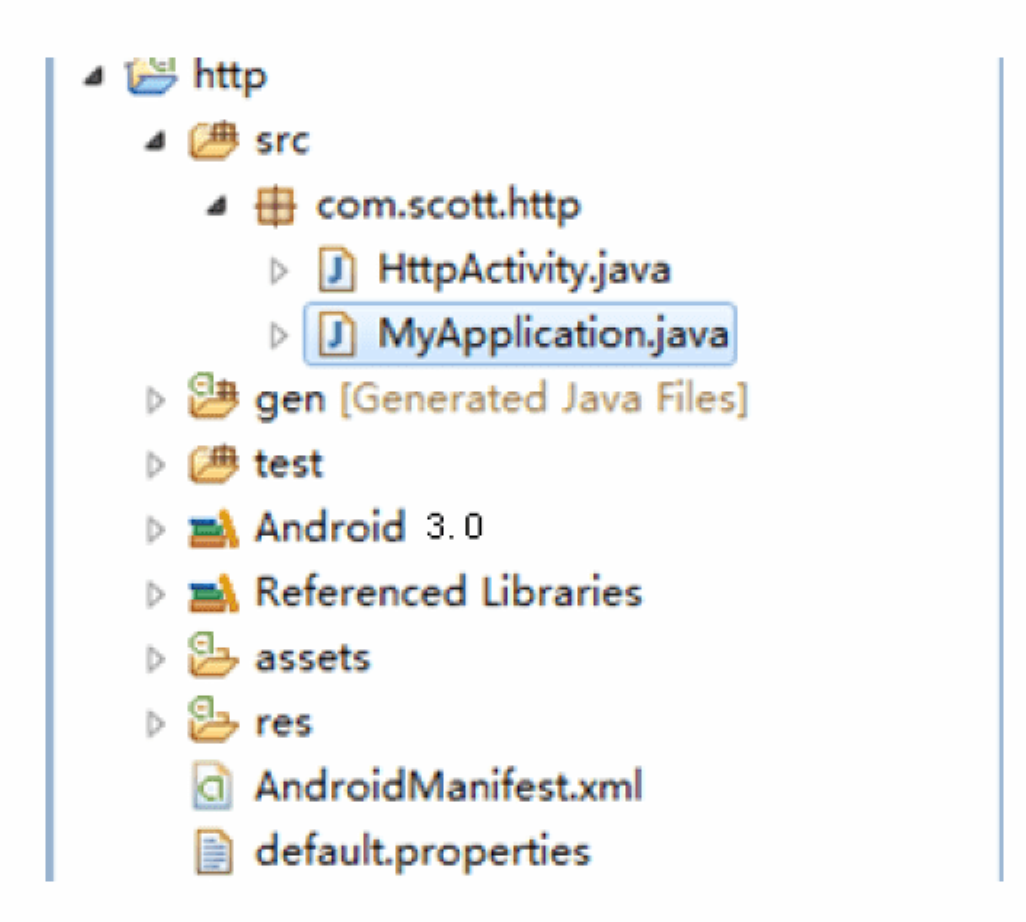


图 3-9 工程文件结构

③ 在文件 MyApplication.java 中扩展了系统的 Application，具体代码如下：

```
package com.scott.http;

import org.apache.http.HttpVersion;
import org.apache.http.client.HttpClient;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.PlainSocketFactory;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.params.HttpParams;
import org.apache.http.params.HttpProtocolParams;
import org.apache.http.protocol.HTTP;

import android.app.Application;

public class MyApplication extends Application {

    private HttpClient httpClient;

    @Override
    public void onCreate() {
        super.onCreate();
        httpClient = this.createHttpClient();
    }

    @Override
    public void onLowMemory() {
        super.onLowMemory();
        this.shutdownHttpClient();
    }
```

```
    @Override
    public void onTerminate() {
        super.onTerminate();
        this.shutdownHttpClient();
    }

    //创建 HttpClient 实例
    private HttpClient createHttpClient() {
        HttpParams params = new BasicHttpParams();
        HttpProtocolParams.setVersion(params, HttpVersion.HTTP 1 1);
        HttpProtocolParams.setContentCharset(params,
            HTTP.DEFAULT CONTENT CHARSET);
        HttpProtocolParams.setUseExpectContinue(params, true);

        SchemeRegistry schReg = new SchemeRegistry();
        schReg.register(new Scheme("http",
            PlainSocketFactory.getSocketFactory(), 80));
        schReg.register(new Scheme("https",
            SSLSocketFactory.getSocketFactory(), 443));

        ClientConnectionManager connMgr = new
            ThreadSafeClientConnManager(params, schReg);

        return new DefaultHttpClient(connMgr, params);
    }

    //关闭连接管理器并释放资源
    private void shutdownHttpClient() {
        if (httpClient != null && httpClient.getConnectionManager() != null) {
            httpClient.getConnectionManager().shutdown();
        }
    }

    //对外提供 HttpClient 实例
    public HttpClient getHttpClient() {
        return httpClient;
    }
}
```

在上述代码中重写了方法 onCreate()，在系统启动时就创建一个 HttpClient；重写了 onLowMemory()和 onTerminate()方法，在内存不足和应用结束时关闭连接，释放资源。需要注意的是，当实例化 DefaultHttpClient 时，传入一个由 ThreadSafeClientConnManager 创建的 ClientConnectionManager 实例，负责管理 HttpClient 的 HTTP 连接。

④ 在文件 AndroidManifest.xml 中进行如下配置，目的是让“优化”版的 Application 生效。

```
<application  android:name=".MyApplication" ...>
…
</application>
```



如果不进行上述配置，系统依旧会默认使用 android.app.Application。在添加上述配置后，系统就会使用前面编写的 com.scott.http.MyApplication，然后就可以在 context 中调用 getApplication()来获取 MyApplication 实例。

经过上面的“优化”处理配置，接下来就可以在活动中应用了。编写的文件 HttpActivity.java 的实现代码如下：

```
package com.scott.http;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;

import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class HttpActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button btn = (Button) findViewById(R.id.btn);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                execute();
            }
        });

    }

    private void execute() {
        try {
            MyApplication app = (MyApplication) this.getApplication();
            //获取 MyApplication 实例
            HttpClient client = app.getHttpClient(); //获取 HttpClient 实例
            HttpGet get = new HttpGet ("http://192.168.1.57:8080/
                web/TestServlet?id=1001&name=john&age=60");
            HttpResponse response = client.execute(get);
            if (response.getStatusLine().getStatusCode() ==
            HttpStatus.SC OK) {
            InputStream is = response.getEntity().getContent();
            String result = inStream2String(is);
```

```
            Toast.makeText(this, result, Toast.LENGTH LONG).show();
        }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //将输入流转换成字符串
    private String inStream2String(InputStream is) throws Exception {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buf = new byte[1024];
        int len = -1;
        while ((len = is.read(buf)) != -1) {
            baos.write(buf, 0, len);
        }
        return new String(baos.toByteArray());
    }
}
```

此时执行后在手机屏幕中单击 execute 按钮后会显示 GET_SUCCESS 的提示，如图 3-10 所示。

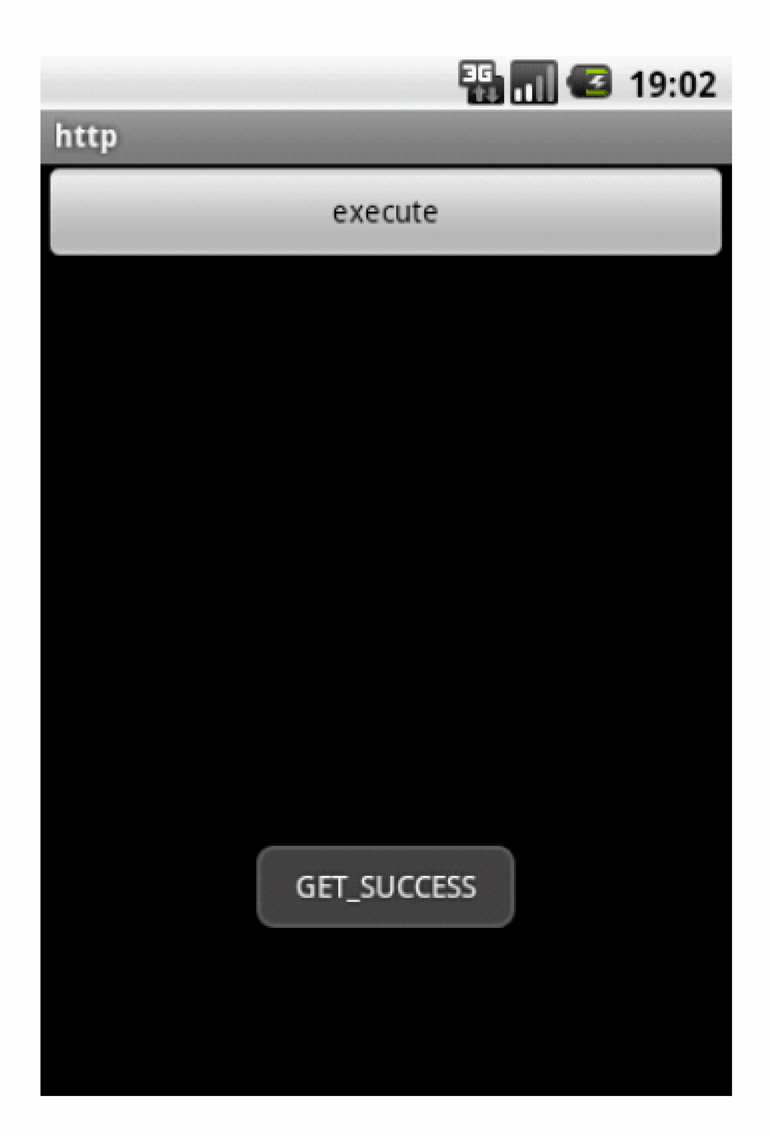


图 3-10　执行效果

3.4　使用标准 Java 接口

本节将带领读者漫游 java.net 包，按照网络方面的知识来逐步学习 Android 中的 Java 网络编程。在讲解过程中穿插了一些有用的演示代码，帮助大家加深对各个知识点的理解。

3.4.1　IP地址

所谓 IP 地址就是给每个连接在 Internet 上的主机分配的一个 32 位地址。java.net 中处理 IP 地址的类是 InetAddress，其结构如图 3-11 所示。

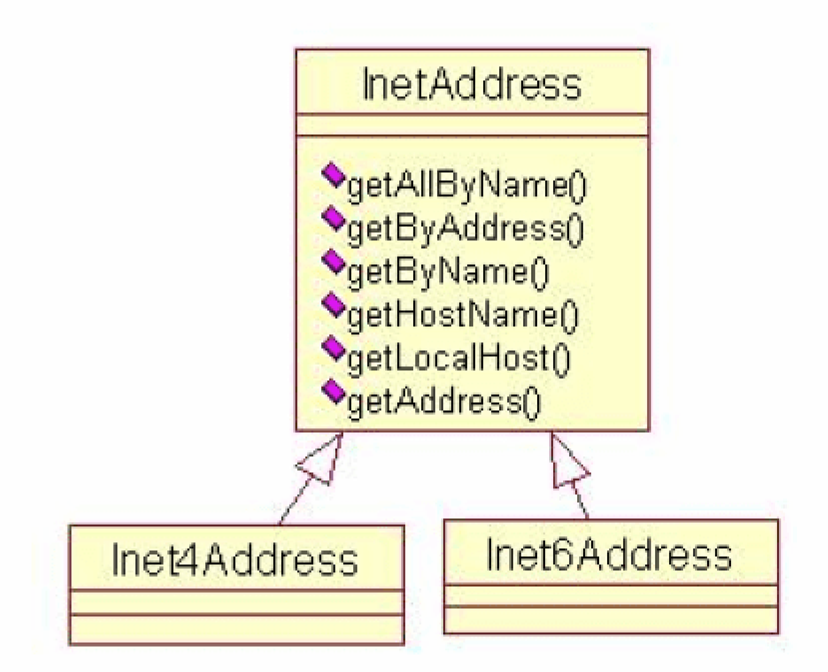


图 3-11　InetAddress结构

下面的代码演示了 InetAddress 的具体用法：

```
String GetHostAddress (String strHostName)
{
InetAddress address = null;
try
{
address = InetAddress.getByName (strHostName);
}
catch(UnknownHostException e)
{
System.out.println(e.getMessage());
}
return InetAddress.getHostAddress () ;
}

void GetAllIP (String strHostName)
{
InetAddress[] add = null;
try
{
add = InetAddress.getAllByName (strHostName);
for(int i=0;i<addr.lenth;i++)
System.out.println(addr[i]);
}
catch(UnknownHostException e)
{
System.out.println(e.getMessage());
}
}
```

上述代码非常简单，但是有一点需要说明：在网络编程时，必须注意异常的捕获。网

络异常是比较正常的现象，比如说当前网络繁忙，网络连接超时更是“家常便饭”。因此在网络编程时，必须养成捕获异常的好习惯，查看完函数说明后，要注意网络异常的说明。特别注意，在使用 getByAddress()函数的时候就必须捕获 UnknownHostException 异常。

3.4.2 套接字Socket类

套接字 Socket 类的基本结构如图 3-12 所示。

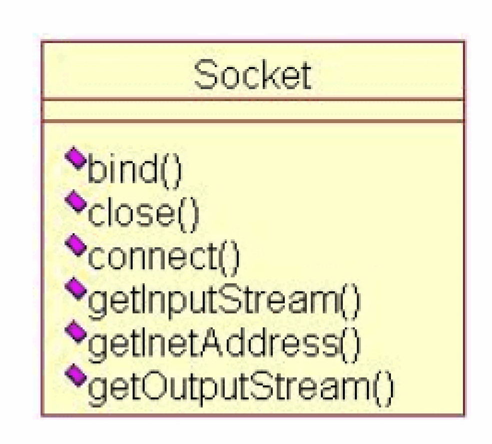


图 3-12 Socket结构

套接字通信的基本思想比较简单，客户端建立一个到服务器的连接，一旦连接建立了，客户端就可以往套接字中写入数据，并向服务器发送数据；反过来，服务器读取客户端写入套接字中的数据。就这么简单，也许细节会复杂些，但是基本思想就如此。看下面一段使用 Socket 类的代码：

```
void WebPing (String strURL)
{
try
{
InetAddress addr;
Socket sock = new Socket(strURL,80);
Addr = sock.getInetAddress ();
System.out.println("Connceted to"+addr);
Sock.close();
}
catch(IOException e)
{
System.out.println(e.getMessage());
}
}
```

如果使用本地主机(localhost)来测试这个程序，则输出如下结果：

```
Connceted to localhost/127.0.0.1
```

其中 InetAddress.toString()的隐含调用(println 调用)自动输出主机名和 IP 地址。另外还有其他套接字，例如 DatagramSocket(通过 UDP 通信的套接字)、MulticastSocket(一种用于多点传送的套接字)以及 ServerSocket(一种用于监听来自客户端的连接的套接字)，在这里就不再一一说明。

3.5　使用 Android 网络接口

在 Android 平台中，可以使用 Android 网络接口 android.net.http 来处理 HTTP 请求。android.net.http 是 android.net 中的一个包，主要包含处理 SSL 证书的类。

3.5.1　android.net.http中的类

在 android.net.http 中存在以下 4 个类。

- AndroidHttpClient
- SslCertificate
- SslCertificate.DName
- SslError

其中 AndroidHttpClient 就是用来处理 HTTP 请求的。

android.net.*实际上是通过对 Apache 的 HttpClient 的封装来实现的一个 HTTP 编程接口，同时还提供了 HTTP 请求队列管理，以及 HTTP 连接池管理，以提高并发请求情况下(如转载网页时)的处理效率，除此之外还有网络状态监视等接口。

下面是一个通过 AndroidHttpClient 访问服务器的简单例子：

```
import import android.net.http.AndroidHttpClient;
     try {
        AndroidHttpClient client = AndroidHttpClient.newInstance
           ("your user agent");
        // 创建 HttpGet 方法，该方法会自动处理 URL 地址的重定向
        HttpGet httpGet = new HttpGet ("http://www.test test.com/");
        HttpResponse response = client.execute(httpGet);
        if (response.getStatusLine().getStatusCode() != HttpStatus.SC_OK) {
           // 错误处理
        }
        // 关闭连接
        client.close();
     } catch (Exception ee) {
     }
```

另外当我们的应用需要同时从不同的主机获取数目不等的数据，并且仅关心数据的完整性而不关心其先后顺序时，也可以使用这部分的接口。典型用例就是 android.webkit 在转载网页和下载网页资源时，具体可参考 android.webkit.*中的相关类来实现。

3.5.2　在手机屏幕中传递HTTP参数

通过前面的学习，了解到 HTTP 是一种网络传输协议，现实中的大多数网页都是通过“HTTP://WWW.”的形式实现显示的。在具体应用时，一些需要的数据都是通过其参数传递的。本节将通过一个具体实例来讲解在手机屏幕中传递 HTTP 参数的流程。

实　例	功　能	源码路径
实例 3-4	在手机屏幕中传递 HTTP 参数	下载路径:\daima\3\httpSHI

1. 实现思路

和网络 HTTP 有关的是 HTTP protocol，在 Android SDK 中，集成了 Apache 的 HttpClient 模块。通过这些模块，可以方便地编写和 HTTP 有关的程序。在 Android SDK 中通常使用 HttpClient 4.0。

在本实例中插入了两个按钮，一个用于以 Post 方式获取网站数据，另外一个用于以 Get 方式获取数据，并以 TextView 对象来显示由服务器端的返回网页内容来显示连接结果。当然首先得建立和 HTTP 的连接，连接之后才能获取 Web Server 返回的结果。

2. 具体实现

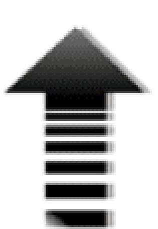
(1) 编写布局文件 main.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:background="@drawable/white"
  android:orientation="vertical"
  android:layout width="fill parent"
  android:layout height="fill parent"
  >
  <TextView
    android:id="@+id/myTextView1"
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:text="@string/title"/>
  <Button
    android:id="@+id/myButton1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="@string/str button1" />
    <Button
    android:id="@+id/myButton2"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="@string/str_button2" />
</LinearLayout>
```

(2) 编写文件 httpSHI.java，其具体实现流程如下。

① 引用 apache.http 相关类实现 HTTP 联机，然后引用 java.io 与 java.util 相关类来读写档案。具体代码如下：

```
/*引用 apache.http 相关类来建立 HTTP 联机*/
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
```



```
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;
import org.apache.http.util.EntityUtils;
/*必须引用java.io 与java.util 相关类来读写档案*/
import irdc.httpSHI.R;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

② 使用 OnClickListener 来侦听单击第一个按钮事件，声明网址字符串并使用建立 Post 方式联机，最后通过 mTextView1.setText 输出提示字符。具体代码如下：

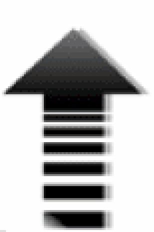
```
/*设定 OnClickListener 来侦听 OnClick 事件*/
mButton1.setOnClickListener(new Button.OnClickListener()
{
  /*覆写 onClick 事件*/
  @Override
  public void onClick(View v)
  {
    /*声明网址字符串*/
    String uriAPI = "http://www.dubblogs.cc:8751/Android/Test/API/
        Post/index.php";
    /*建立 HTTP  Post 联机*/
    HttpPost httpRequest = new HttpPost(uriAPI);
    /*
     * Post 运行传送变量必须用 NameValuePair[]数组存储
    */
    List <NameValuePair> params = new ArrayList <NameValuePair>();
    params.add(new BasicNameValuePair("str", "I am Post String"));
    try
    {
      httpRequest.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF 8));
      /*取得 HTTP 输出*/
      HttpResponse httpResponse = new DefaultHttpClient().execute(httpRequest);
      /*如果状态码为 200 */
      if(httpResponse.getStatusLine().getStatusCode() == 200)
      {
        /*获取应答字符串*/
        String strResult = EntityUtils.toString(httpResponse.getEntity());
        mTextView1.setText(strResult);
```

```
      }
      else
      {
        mTextView1.setText("Error Response:
          "+httpResponse.getStatusLine().toString());
      }
    }
    catch (ClientProtocolException e)
    {
      mTextView1.setText(e.getMessage().toString());
      e.printStackTrace();
    }
    catch (IOException e)
    {
      mTextView1.setText(e.getMessage().toString());
      e.printStackTrace();
    }
    catch (Exception e)
    {
      mTextView1.setText(e.getMessage().toString());
      e.printStackTrace();
    }

  }
});
```

③ 使用 OnClickListener 来侦听单击第二个按钮的事件，声明网址字符串并建立 Get 方式的联机功能，分别实现发出 HTTP 获取请求、获取应答字符串和删除冗余字符操作，最后通过 mTextView1.setText 输出提示字符。具体代码如下：

```
mButton2.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub
    /*声明网址字符串*/
    String uriAPI = "http://www.XXXX.cc:8751/index.php?str=I+am+Get+String";
    /*建立 HTTP Get 联机*/
    HttpGet httpRequest = new HttpGet(uriAPI);
    try
    {
      /*发出 HTTP 获取请求*/
      HttpResponse httpResponse = new DefaultHttpClient().execute(httpRequest);
      /*若状态码为 200 ok*/
      if(httpResponse.getStatusLine().getStatusCode() == 200)
      {
        /*获取应答字符串*/
        String strResult = EntityUtils.toString(httpResponse.getEntity());
        /*删除冗余字符*/
        strResult = eregi_replace("(\r\n|\r|\n|\n\r)","",strResult);
```



```
            mTextView1.setText(strResult);
          }
          else
          {
            mTextView1.setText("Error Response:
               "+httpResponse.getStatusLine().toString());
          }
        }
        catch (ClientProtocolException e)
        {
          mTextView1.setText(e.getMessage().toString());
          e.printStackTrace();
        }
        catch (IOException e)
        {
          mTextView1.setText(e.getMessage().toString());
          e.printStackTrace();
        }
        catch (Exception e)
        {
          mTextView1.setText(e.getMessage().toString());
          e.printStackTrace();
        }
      }
    });
  }
```

④ 定义替换字符串函数 eregi_replace 来替换掉一些非法字符，具体代码如下：

```
    /* 字符串替换函数 */
    public String eregi replace(String strFrom, String strTo, String strTarget)
    {
      String strPattern = "(?i)"+strFrom;
      Pattern p = Pattern.compile(strPattern);
      Matcher m = p.matcher(strTarget);
      if(m.find())
      {
        return strTarget.replaceAll(strFrom, strTo);
      }
      else
      {
        return strTarget;
      }
    }
}
```

(3) 在文件 AndroidManifest.xml 中声明网络连接权限，具体代码如下：

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

执行后的效果如图 3-13 所示，单击图中的按钮能够以不同方式获取 HTTP 参数。

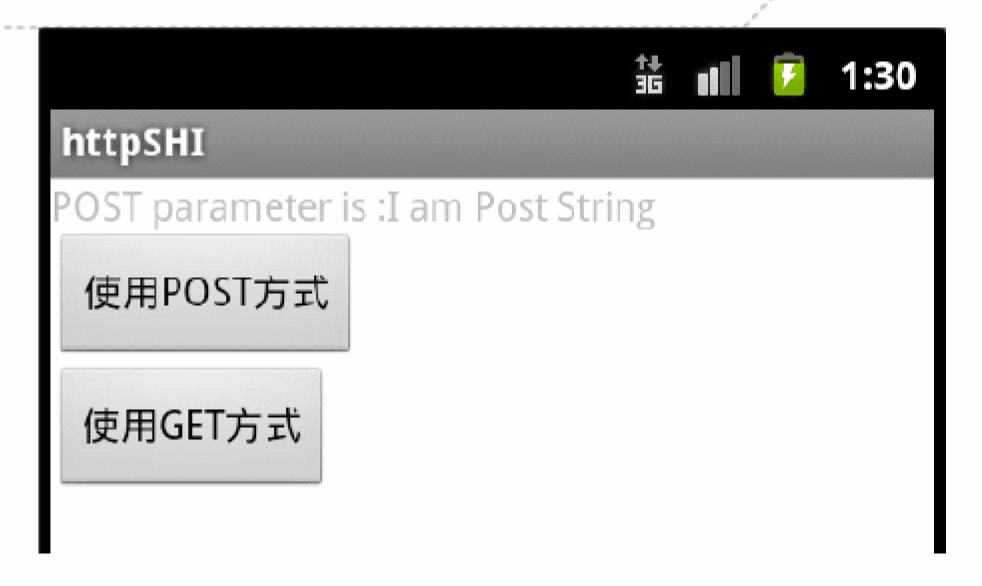


图 3-13　单击“使用PSST方式”按钮后的效果

注意： 在 Android 系统中打开连接的通用代码如下：

```
Intent it = new Intent(Intent.ACTION VIEW,
    Uri.parse("http://www.baidu.com"));
it.setClassName("com.Android.browser",
    "com.android.browser.BrowserActivity");
getContext().startActivity(it);
```

在 Android 系统中打开本地网页的通用代码如下：

```
Intent intent=new Intent();
intent.setAction("android.intent.action.VIEW");
Uri CONTENT URI BROWSERS = Uri.parse
    ("content://com.android.htmlfileprovider/sdcard/123.html");
intent.setData(CONTENT URI BROWSERS);
intent.setClassName("com.android.browser",
    "com.android.browser.BrowserActivity");
startActivity(intent);
```

第 4 章

URL 处理

在网络中访问一个网页时，需要通过 URL 地址来指定将要访问的页面。在互联网中，URL 是我们访问 Web 页面的地址。基于 URL 的重要性，所以本书将用一章的内容来讲解在 Android 系统中处理 URL 的基本知识。

4.1 使用 URL 类

URL 是 Uniform Resource Locator 的缩写，意为统一资源定位器，它是指向互联网“资源”的指针。资源可以是简单的文件或目录，也可以是对更为复杂的对象引用，例如对数据库或搜索引擎的查询。通常 URL 可以由协议名、主机、端口和资源组成。即满足如下格式：

```
protocol://host:port/resourceName
```

例如下面的 URL 地址：

```
http://www.oneedu.cn/Index.htm
```

在 Android 系统中可以通过 URL 获取网络资源，其中 URLConnection 和 HttpURLConnection 是最为常用的两种方式。

4.1.1 URL类基础

在 Java 应用中，为我们提供了类 URL 来处理和 URL 相关的知识，此类的具体结构如图 4-1 所示。

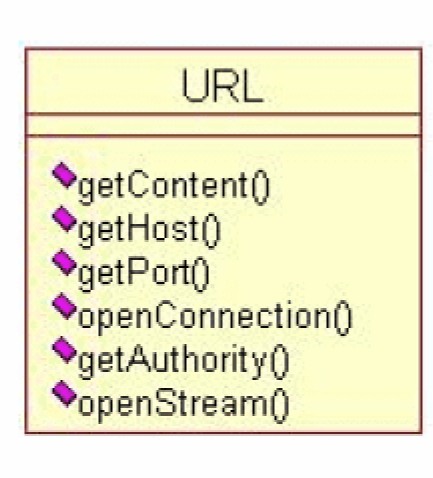


图 4-1　URL 结构

类 URL 被包含在 Java.net 包中，此类是 Java 实现网络编程应用的重要内容。类 URL 为 Java 访问网络资源提供了接口，通过这些接口可以很容易地访问服务器上的文件。类 URL 有以下四种常用的构造方法格式。

(1) public URL(String spec)：其功能是通过一个表示 URL 地址的字符串构造一个 URL 对象，例如：

```
URL a=new URL("http://www.baidu.com")
```

(2) public URL(URL context, String spec)：其功能是通过基本 URL 和相对 URL 指定文件构造一个 URL 对象，例如：

```
URL b=new URL("index.jsp")
```

(3) public URL(String protocol,String host,String file)：其功能是通过协议名、主机名和相对的 URL 指定文件构造一个 URL 对象，例如：

```
URL c= new URL("http", "www.sina.com.cn", "download/index.html")
```

(4) public URL(String protocol,String host,int port,String file)：其功能是通过协议名、主机名和端口号和相对的 URL 指定文件构造一个 URL 对象，例如：

```
URL d= new URL("http", "www.sina.com.cn", "6870","download/index.html")
```

实　例	功　能	源码路径
实例 4-1	演示异常处理 URL 的方法	下载路径:\daima\4\WangURL.java

在构造 URL 程序时，经常会发生 MalformedURLException 异常，此时需要在程序中设置异常处理。通过本实例的演示代码，讲解 URL 处理异常的基本知识。

```
import java.io.*;
import java.net.*;
public class WangURL
{
    public static void main(String args[])
    {
        try
       {
        URL u1=new URL("http://www.baidu.com/");
       }
       catch(MalformedURLException e)
       {
           System.out.println(e);
       }
       catch(IOException ee)
       {
           System.out.println(ee);
       }
       catch(Exception eee)
       {
           System.out.println(eee);
       }
    }
}
```

执行上述程序，在网络没有错误的情况下不会有任何执行效果。在 URL 类中有很多属性，例如协议名、主机名和端口号等，当对象产生后，其属性是不能改变的。在 URL 类的 API 中定义了很多方法来获得这些属性，下面列出了一些经常用到的方法。

- public String getProtocol()：获取该 URL 的协议名。
- public String getHost()：获取该 URL 的主机名。
- public int getPort()：获取该 URL 的端口号。
- public String getFiel()：获取该 URL 的文件名。
- public String getRef()：获取该 URL 在文件中的相对位置。
- public String getQuery()：获取该 URL 的路径。
- public String getPath()：获取该 URL 的路径。
- public String getAuthority()：获取该 URL 的权限信息。
- public String getUserInfo()：获得该 URL 的锚。

使用上述方法的具体过程非常简单，下面将通过一个具体实例来讲解使用 API 方法的过程。

实　例	功　能	源码路径
实例 4-2	获得端口号和 URL 的文件名	下载路径:\daima\4\URL1.java

通过本实例的演示代码，讲解如何获得对方端口号和指定 URL 文件名的过程。文件

URL1.java 的实现代码如下：

```
import java.io.*;
import java.net.*;
public class URL1
{
    public static void main(String args[])
    {
        try
       {
        URL u1=new URL("http://mil.news.sohu.com/");
          //获取该 URL 的协议名
        System.out.println(u1.getProtocol());
          //获取该 URL 的主机名
        System.out.println(u1.getHost());
          //获取该 URL 的端口号
        System.out.println(u1.getPort());
          //获取该 URL 的文件名
        System.out.println(u1.getFile());
       }
       catch(MalformedURLException e)
       {
           System.out.println(e);
       }
       catch(IOException ee)
       {
           System.out.println(ee);
       }
       catch(Exception eee)
       {
           System.out.println(eee);
       }
    }
}
```

执行上述代码后的效果如图 4-2 所示。

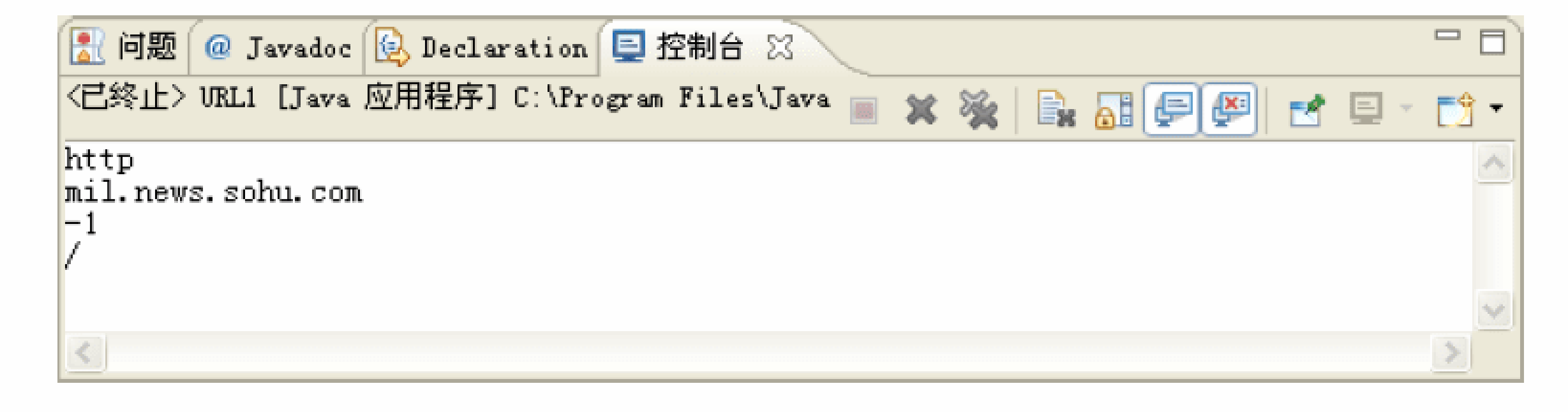


图 4-2　执行效果

在实例 4-2 中，讲解了获取 URL 的一些常用信息，下面展示一段代码，将前面讲解的 I/O 和这个知识点结合，读者将代码从光盘复制到自己的电脑中，进行编译，查看结果。其代码如下：

```
import java.io.*;
import java.net.*;
```



```
public class URL2
{
    public static void main(String args[])
    {
        try
        {
            //创建一个 URL 对象
            URL u1=new URL("http://www.163.com/");
            //构造一个 BufferedReader 对象
            BufferedReader br=new BufferedReader(
                      new InputStreamReader(u1.openStream()));
            String s;
            //从输入流不停地读数据，直到读完为止
            while((s=br.readLine())!=null)
            {
                //把读入的数据打印出来
                System.out.println(s);
            }
            //关闭输入流
            br.close();
        }
        catch(MalformedURLException e)
        {
            System.out.println(e);
        }
        catch(IOException ee)
        {
            System.out.println(ee);
        }
        catch(Exception eee)
        {
            System.out.println(eee);
        }
    }
}
```

4.1.2　URI和URL的使用

在 JDK 中还提供了一个名为 URI 的类，URI 是 Uniform Resource Identifiers 的缩写。URI 的实例代表一个统一资源标识符，Java 中的 URI 不能用于定位任何资源，其唯一作用就是解析。与此相对应的是，URL 包含了一个可打开到达该资源的输入流，因此可以将 URL 理解成 URI 的特例。

在类 URL 中提供了多个构造器用于创建 URL 对象，一旦获得了 URL 对象后，就可以调用下面的方法来访问该 URL 对应的资源。

- String getFile()：获取此 URL 的资源名。
- String getHost()：获取此 URL 的主机名。
- String getPath()：获取此 URL 的路径部分。

- int getPort()：获取此 URL 的端口号。
- String getProtocol()：获取此 URL 的协议名称。
- String getQuery()：获取此 URL 的查询字符串部分。
- URLConnection openConnection()：返回一个 URLConnection 对象，它表示到 URL 所引用的远程对象的连接。
- InputStream openStream()：打开与此 URL 的连接，并返回一个用于读取该 URL 资源的 InputStream。

了解了上述 URL 类的基本知识后，接下来将通过几段演示代码来简要介绍使用类 URL 的基本知识。

1. 使用URL实现多线程下载

在上面列出的前几个方法都非常容易理解，可以使用其中的方法 openStream()读取该 URL 资源的 InputStream(输入流)，通过该方法可以非常方便地读取远程资源，甚至实现多线程下载。例如下面的演示代码：

```
class DownThread extends Thread
{
//定义字节数组的长度
private final int BUFF_LEN = 32;
//定义下载的起始点
private long start;
//定义下载的结束点
private long end;
//下载资源对应的输入流
private InputStream is;
// 将下载到的字节输出到 raf 中
private RandomAccessFile raf ;
//构造器，传入输入流、输出流和下载起始点、结束点
public DownThread(long start , long end
, InputStream is , RandomAccessFile raf)
{
//输出该线程负责下载的字节位置
System.out.println(start + "---->"  + end);
this.start = start;
this.end = end;
this.is = is;
this.raf = raf;
}
public void run()
{
try
{
is.skip(start);
raf.seek(start);
//定义读取输入流内容的缓存数组
byte[] buff = new byte[BUFF_LEN];
//本线程负责下载资源的大小
long contentLen = end - start;
```



```
//定义最多需要读取几次就可以完成本线程的下载
long times = contentLen / BUFF LEN + 4;
//实际读取的字节数
int hasRead = 0;
for (int i = 0; i < times ; i++)
{
hasRead = is.read(buff);
//如果读取的字节数小于 0，则退出循环！
if (hasRead < 0)
{
break;
}
raf.write(buff , 0 , hasRead);
}
}
catch (Exception ex)
{
ex.printStackTrace();
}
// 使用 finally 块来关闭当前线程的输入流、输出流
finally
{
try
{
if (is != null)
{
is.close();
}
if (raf != null)
{
raf.close();
}
}
catch (Exception ex)
{
ex.printStackTrace();
}
}
}
}
public class MutilDown
{
public static void main(String[] args)
{
final int DOWN THREAD NUM = 4;
final String OUT FILE NAME = "down.jpg";
InputStream[] isArr = new InputStream[DOWN THREAD NUM];
RandomAccessFile[] outArr = new RandomAccessFile[DOWN THREAD NUM];
try
{
//创建一个 URL 对象
```

```
URL url = new URL("http://images.china-pub.com/+"ebook35001-
            40000/35850/shupi.jpg");
//以此 URL 对象打开第一个输入流
isArr[0] = url.openStream();
long fileLen = getFileLength(url);
System.out.println(" 网络资源的大小" + fileLen);
//以输出文件名创建第一个 RandomAccessFile 输出流
outArr[0] = new RandomAccessFile(OUT FILE NAME , "rw");
//创建一个与下载资源相同大小的空文件
for (int i = 0 ; i < fileLen ; i++ )
{
outArr[0].write(0);
}
// 每线程应该下载的字节数
long numPerThred = fileLen / DOWN THREAD NUM;
//整个下载资源整除后剩下的余数
long left = fileLen % DOWN THREAD NUM;
for (int i = 0 ; i < DOWN THREAD NUM; i++)
{
//为每个线程打开一个输入流、一个 RandomAccessFile 对象,
// 让每个线程分别负责下载资源的不同部分。
if (i != 0)
{
//以 URL 打开多个输入流
isArr = url.openStream();
//以指定输出文件创建多个 RandomAccessFile 对象
outArr = new RandomAccessFile(OUT FILE NAME , "rw");
}
//分别启动多个线程来下载网络资源
if (i == DOWN THREAD NUM - 1 )
{
//最后一个线程下载指定 numPerThred+left 个字节
new DownThread(i * numPerThred , (i + 1) * numPerThred + left, isArr ,
    outArr).start();
}
else
{
//每个线程负责下载一定的 numPerThred 个字节
new DownThread(i * numPerThred , (i + 1) * numPerThred,
isArr , outArr).start();
}
}
}
catch (Exception ex)
{
ex.printStackTrace();
}
}
//定义获取指定网络资源的长度的方法
public static long getFileLength(URL url) throws Exception
{
```



```
long length = 0;
//打开该 URL 对应的 URLConnection
URLConnection con = url.openConnection();
// 获取连接 URL 资源的长度
long size = con.getContentLength();
length = size;
return length;
}
}
```

在上面的代码中，定义了一个名为 DownThread 线程类，该线程从 start(开始)InputStream 中读取数据，到 end(结束)时完成所有字节数据的读取工作，并写入 RandomAccessFile 对象。这个 DownThread 线程类的方法 run()就是一个简单的输入、输出实现。

在上述代码中的 MutilDown 类中，方法 main()按以下步骤来实现多线程下载。

(1) 创建 URL 对象。

(2) 获取指定 URL 对象所指向资源的大小(由 getFileLength 方法实现)，此处用到了 URLConnection 类，该类代表 Java 应用程序和 URL 之间的通信连接。

(3) 在本地磁盘上创建一个与网络资源相同大小的空文件。

(4) 计算每条线程应该下载网络资源的哪个部分(从哪个字节开始，到哪个字节结束)。

(5) 依次创建、启动多条线程来下载网络资源的指定部分。

上面程序已经实现了多线程下载的核心代码，如果要实现断点下载，则还需要额外增加一个配置文件，该配置文件分别记录每个线程已经下载到了哪个字节，当网络断开后再次开始下载时，每个线程根据配置文件里记录的位置向后下载即可。

在 URL 中，可以通过方法 openConnection()返回一个 URLConnection 对象，该对象表示应用程序和 URL 之间的通信连接。程序可以通过 URLConnection 实例向该 URL 发送请求、读取 URL 引用的资源。

创建一个和 URL 的连接，并发送请求、读取此 URL 引用资源的基本步骤如下。

(1) 通过调用 URL 对象 openConnection()方法来创建 URLConnection 对象。

(2) 设置 URLConnection 的参数和普通请求属性。

(3) 如果只是发送 Get 方式请求，使用 Connect 方法建立和远程资源之间的实际连接即可；如果需要发送 Post 方式的请求，需要获取 URLConnection 实例对应的输出流来发送请求参数。

(4) 如果远程资源可用，此时程序可以访问远程资源的头字段或通过输入流读取远程资源的数据。

在建立和远程资源的实际连接之前，我们可以通过以下方法来设置请求头字段。

- setAllowUserInteraction：设置该 URLConnection 的 allowUserInteraction 请求头字段的值。
- setDoInput：设置该 URLConnection 的 doInput 请求头字段的值。
- setDoOutput：设置该 URLConnection 的 doOutput 请求头字段的值。
- setIfModifiedSince：设置该 URLConnection 的 ifModifiedSince 请求头字段的值。
- setUseCaches：设置该 URLConnection 的 useCaches 请求头字段的值。

除此之外，还可以使用以下方法来设置或增加通用头字段。

- setRequestProperty(String key, String value)：设置该 URLConnection 的 key 请求头字段的值为 value。例如下面的代码：

```
conn.setRequestProperty("accept" , "*/*")
```

- addRequestProperty(String key, String value)：为该 URLConnection 的 key 请求头字段增加 value 值，该方法并不会覆盖原请求头字段的值，而是将新值追加到原请求头字段中。

当远程资源可用之后，程序可以使用以下方法来访问头字段和内容。

- Object getContent()：获取该 URLConnection 的内容。
- String getHeaderField(String name)：获取指定响应头字段的值。
- getInputStream()：返回该 URLConnection 对应的输入流，用于获取 URLConnection 响应的内容。
- getOutputStream()：返回该 URLConnection 对应的输出流，用于向 URLConnection 发送请求参数。

注意： 如果既要使用输入流读取 URLConnection 响应的内容，又要使用输出流发送请求参数，一定要先使用输出流，再使用输入流。

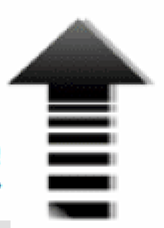
方法 getHeaderField()可以根据响应头字段来返回对应的值。而某些头字段由于经常需要访问，所以 Java 提供以下方法来访问特定响应头字段的值。

- getContentEncoding：获取 content-encoding 响应头字段的值。
- getContentLength：获取 content-length 响应头字段的值。
- getContentType：获取 content-type 响应头字段的值。
- getDate()：获取 date 响应头字段的值。
- getExpiration()：获取 expiration 响应头字段的值。
- getLastModified()：获取 last-modified 响应头字段的值。

2. 使用URL实现和站点的交互

在日常项目应用中，经常需要向 Web 站点发送 Get 请求、Post 请求，并从 Web 站点获得响应。例如我们可以通过如下代码来实现：

```
public class TestGetPost
{
/**
* 向指定 URL 发送 Get 方法的请求
* @param url 发送请求的 URL
* @param param 请求参数，请求参数应该是 name1=value1&name2=value2 的形式
* @return URL 所代表远程资源的响应
*/
public static String sendGet(String url , String param)
{
String result = "";
BufferedReader in = null;
try
```



```
{
String urlName = url + "?" + param;
URL realUrl = new URL(urlName);
//打开和 URL 之间的连接
URLConnection conn = realUrl.openConnection();
//设置通用的请求属性
conn.setRequestProperty("accept", "*/*");
conn.setRequestProperty("connection", "Keep-Alive");
conn.setRequestProperty("user-agent",
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)");
//建立实际的连接
conn.connect();
//获取所有响应头字段
Map<String, List<String>> map = conn.getHeaderFields();
//遍历所有的响应头字段
for (String key : map.keySet())
{
System.out.println(key + "--->" + map.get(key));
}
//定义 BufferedReader 输入流来读取 URL 的响应
in = new BufferedReader(
new InputStreamReader(conn.getInputStream()));
String line;
while ((line = in.readLine())!= null)
{
esult += "n" + line;
}
}
catch(Exception e)
{
System.out.println("发送 GET 请求出现异常！" + e);
e.printStackTrace();
}
//使用 finally 块来关闭输入流
finally
{
try
{
if (in != null)
{
in.close();
}
}
catch (IOException ex)
{
ex.printStackTrace();
}
}
return result;
}
/**
```

```
* 向指定 URL 发送 Post 方法的请求
* @param url 发送请求的 URL
* @param param 请求参数，请求参数应该是 name1=value1&name2=value2 的形式
* @return URL 所代表远程资源的响应
*/
public static String sendPost(String url,String param)
{
PrintWriter out = null;
BufferedReader in = null;
String result = "";
try
{
URL realUrl = new URL(url);
//打开和 URL 之间的连接
URLConnection conn = realUrl.openConnection();
//设置通用的请求属性
conn.setRequestProperty("accept", "*/*");
conn.setRequestProperty("connection", "Keep-Alive");
conn.setRequestProperty("user-agent",
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)");
//发送 Post 请求必须设置如下两行
conn.setDoOutput(true);
conn.setDoInput(true);
// 获取 URLConnection 对象对应的输出流
out = new PrintWriter(conn.getOutputStream());
//发送请求参数
out.print(param);
//flush 输出流的缓冲
out.flush();
//定义 BufferedReader 输入流来读取 URL 的响应
in = new BufferedReader(
new InputStreamReader(conn.getInputStream()));
String line;
while ((line = in.readLine())!= null)
{
result += "n" + line;
}
}
catch(Exception e)
{
System.out.println("发送 POST 请求出现异常！" + e);
e.printStackTrace();
}
//使用 finally 块来关闭输出流、输入流
finally
{
try
{
if (out != null)
{
out.close();
```

```
}
if (in != null)
{
in.close();
}
}
catch (IOException ex)
{
ex.printStackTrace();
}
}
return result;
}
//提供主方法，测试发送 Get 请求和 Post 请求
public static void main(String args[])
{
//发送 Get 请求
String s = TestGetPost.sendGet("http://localhost:8888/abc/
login.jsp",null);
System.out.println(s);
// 发送 Post 请求
String s1 = TestGetPost.sendPost("http://localhost:8888/abc/a.jsp",
"user= 李刚&pass=abc");
System.out.println(s1);
}
}
```

在上述代码中，在发送 Get 请求时只需将请求参数放在 URL 字符串之后用“?”隔开，程序直接调用 URLConnection 对象的 Connect 方法即可。如果程序需要发送 Post 请求，则需要先设置 doIn 和 doOut 两个请求头字段的值，然后再使用 URLConnection 对应的输出流来发送请求参数即可。

不管是发送 Get 请求，还是发送 Post 请求，程序获取 URLConnection 响应的方式完全一样：如果程序可以确定远程响应是字符流，则可以使用字符流来读取；如果程序无法确定远程响应是字符流，则使用字节流读取即可。

4.2　使用 URLConnection 类

在一般情况下，URL 类就可以满足我们的项目需求，但是在一些特殊情况下，比如 HTTP 数据头的传递，这个时候我们就得使用 URLConncetion 类。类 URLConncetion 的结构如图 4-3 所示。

使用 URLConncetion 类后，我们对网络的控制就增加了很多，例如下面的代码。

```
void SendRequest (String strURL)
{
URL url = URL(strURL);
HttpURLConnection conn = (HttpURLConnection)url.openConnection ();
```

```
conn.setDoInput (true);
conn.setDoOutput (true);
conn.setRequestProperty ("Content-type","application/xxx");
conn.connect ();
System.out.println(Conn.getResponseMessage ());
InputMessage is = Conn.getIputStream();
int c;
do{
c = is.read();
if(c!=-1) System.out.println((char)c);
}while(c!=-1)
}
```

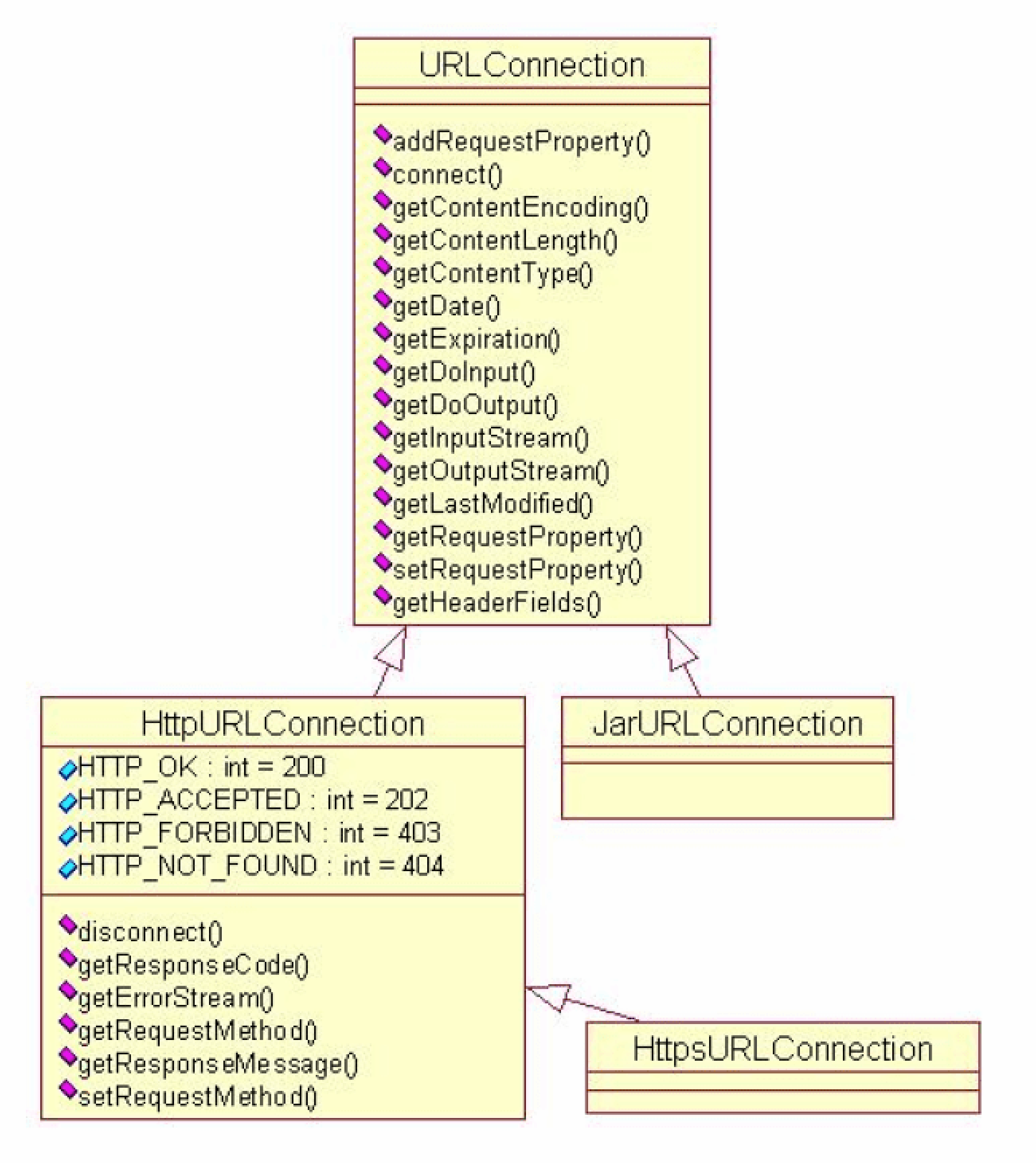


图 4-3　URLConncetion结构

其实在编程时，我们无须担心普通 Java 平台和 Android 平台的差异，我们完全可以使用传统的 Java 知识。在接下来的内容中，将通过具体实例来讲解在 Android 中使用 URLConnection 的基本流程。

1. 在手机屏幕中显示QQ空间中的照片

网络真是无奇不有，在 QQ 空间中可以存放我们的照片。

实　例	功　能	源码路径
实例 4-3	在手机屏幕中显示 QQ 空间中的照片	下载路径:\daima\4\QQ

在本实例中，直接在 Gallery 中显示 QQ 空间中的照片，这样可以节约手机的存储空间。在具体实现上，需要将 URL 网址的照片实时处理下载后，以 InputStream 转换为



Bitmap，这样才能放入 BaseAdapter 中取用。在运行实例前，需要预先准备照片并上传到网络空间中，在获取照片的连接后，再以 String 数组方式放在程序中，并对 BaseAdapter 稍作修改，加上 URL 对象的访问以及 URLConnection 连接的处理。

(1) 编写布局文件 main.xml，在里面插入了一个 Gallery 控件来实现滑动照片效果。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/myLinearLayout"
  android:orientation="vertical"
  android:layout width="fill parent"
  android:layout height="fill parent"
  >
  <Gallery
  android:id="@+id/myGallery01"
  android:layout width="fill parent"
  android:layout height="fill parent">
  </Gallery>
</LinearLayout>
```

(2) 编写主程序文件 QQ.java，其具体实现流程如下。

① 分别声明在 Gallery 中要显示的 5 张图片的地址栏字符串，具体代码如下：

```
public class QQ extends Activity
{
  private Gallery myGallery01;
  /* 地址栏字符串 */
  private String[] myImageURL = new String[]
  {
     "http://b27.photo.store.qq.com/http imgload.cgi?/"
        + "rurl4 b=086a67cbd6a8cfb4389ea2b48efab6f322f755a085107a7aeeaa56
        fc1358b1bd124186254e021f0655732688e69f060725491f8ae82e8e5508dbe
        9821670e2baf04e92dedc97e3bbf28e5605596aa991c13220f1&a=27&b=27",
     "http://b27.photo.store.qq.com/http imgload.cgi?/"
        + "rurl4 b=086a67cbd6a8cfb4389ea2b48efab6f3ea78f5797abbbaa617259
        f2d2a980a5468f2801897cfcc2b78af92fbb87565ed7a3a08041daff2dd9ccd
        26d3cc6198e41f2d205c8a0c445325771e8a179215999afaf9f3&a=27&b=27",
     "http://b27.photo.store.qq.com/http imgload.cgi?/"
        + "rurl4 b=2a9dcf1fd909a7ed3ce8951f738608982f26d812b3a5fc96e221
        b85fc085e7cc3264ee20730f0fd3a1f7aca06740db7a6153d9357467ca39f82
        b866b6fbe3cd94bbdd10ed01841e67c95d8e4af8890b7ced40869&a=30&b=27",
     "http://b27.photo.store.qq.com/http imgload.cgi?/"
        + "rurl4 b=2a9dcf1fd909a7ed3ce8951f73860898bb7ff57a8cb7747c9f0eb6
        a02124850b709c0b86f086a4ba5653eeb71dd4b01e4a58f407e2eec9433cd8
        d4bc0b88fda56260c2c8beb34ebab77b610c7131393f82e774ef&a=27&b=27",
     "http://b27.photo.store.qq.com/http imgload.cgi?/"
        + "rurl4 b=2a9dcf1fd909a7ed3ce8951f73860898158d252489f84e7d2a83
        d44c01b7bb12b2c19ca0efdd555dba788407fd01e9de45524b11a9793f53262
        4197bc8d14c84ae78ddebafe4357e4eedc60e9e510224367490bf&a=27&b=27" };
```

② 引入布局文件 main.xml，定义类成员 myContext Context 对象，然后设置只有一个

参数 C 的构造器。具体代码如下：

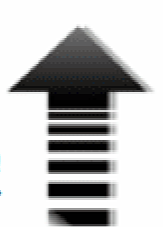
```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  myGallery01 = (Gallery) findViewById(R.id.myGallery01);
  myGallery01.setAdapter(new myInternetGalleryAdapter(this));
}
/* 用 BaseAdapter */
public class myInternetGalleryAdapter extends BaseAdapter
{
  /* 类成员 myContext Context 对象 */
  private Context myContext;
  private int mGalleryItemBackground;
  /*构造器只有一个参数，即要存储的 Context */
  public myInternetGalleryAdapter(Context c)
  {
    this.myContext = c;
    TypedArray a = myContext
        .obtainStyledAttributes(R.styleable.Gallery);
    /*   获取 Gallery 属性的 Index id */
    mGalleryItemBackground = a.getResourceId(
        R.styleable.Gallery android galleryItemBackground, 0);
    /* 把对象的 styleable 属性能够反复使用 */
    a.recycle();
  }
```

③ 定义方法 getCount()来返回全部已定义照片的总量，定义方法 getItem(int position)获取当前容器中照片数的数组 ID。具体代码如下：

```
  /*   返回全部已定义照片的总量 */
  public int getCount()
  {
    return myImageURL.length;
  }
  /* 使用 getItem 方法获取当前容器中照片数的数组 ID */
  public Object getItem(int position)
  {
    return position;
  }
  public long getItemId(int position)
  {
    return position;
  }
```

④ 定义方法 getScale，利用 getScale 根据中央位移量返回 views 的大小。具体代码如下：

```
  /* 根据中央位移量，利用 getScale 返回 views 的大小(0.0f to 1.0f) */
  public float getScale(boolean focused, int offset)
  {
```



```
    /* Formula: 1 / (2 ^ offset) */
    return Math.max(0, 1.0f / (float) Math.pow(2, Math
        .abs(offset)));
  }
```

执行后将在 Gallery 中显示指定的照片，如图 4-4 所示。

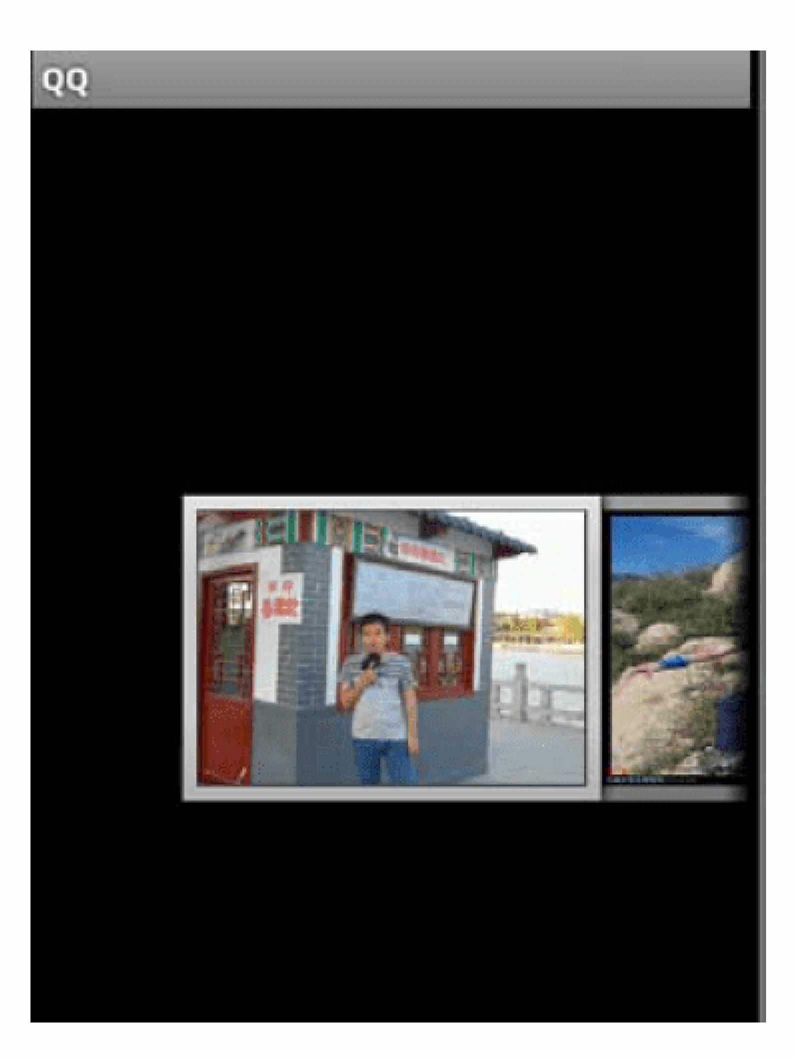


图 4-4 执行效果

2. 从网络中下载图片作为屏幕背景

我们可以从网络中下载一个图片文件来作为手机屏幕的背景。

实 例	功 能	源码路径
实例 4-4	从网络中下载图片作为屏幕背景	下载路径:\daima\4\pingmu

本实例的功能是远程获取网络中的一张图片，并将该图片作为手机屏幕的背景。当下载完图片后，通过 InputStream 传到 ContextWrapper 中重写 setWallpaper 的方式实现的。其中传入的参数是 URLConnection.getInputStream()中的数据内容。本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，分别插入一个文本框控件和按钮控件。主要代码如下：

```
<EditText
  android:id="@+id/myEdit"
  android:layout width="280px"
  android:layout height="wrap content"
  android:text="http://"
  android:textSize="12sp"
  android:layout x="20px"
  android:layout y="42px"
>
</EditText>
<TextView
  android:id="@+id/myText"
  android:layout_width="wrap_content"
```

```
  android:layout height="wrap content"
  android:text="@string/str title"
  android:textSize="16sp"
  android:textColor="@drawable/black"
  android:layout_x="20px"
  android:layout y="12px"
>
</TextView>
<Button
  android:id="@+id/myButton1"
  android:layout width="80px"
  android:layout height="45px"
  android:text="@string/str button1"
  android:layout x="70px"
  android:layout y="102px"
>
</Button>
<Button
  android:id="@+id/myButton2"
  android:layout width="80px"
  android:layout height="45px"
  android:text="@string/str button2"
  android:layout x="150px"
  android:layout y="102px"
>
</Button>
<ImageView
  android:id="@+id/myImage"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:layout x="20px"
  android:layout y="152px"
>
</ImageView>
```

(2) 编写主程序文件 pingmu.java，具体实现流程如下。

① 单击 mButton1 按钮时通过 mButton1.setOnClickListener 来预览图片，如果网址为空则输出空白提示，如果不为空则传入“type=1”，表示预览图片。具体代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  /* 初始化对象 */
  mButton1 =(Button) findViewById(R.id.myButton1);
  mButton2 =(Button) findViewById(R.id.myButton2);
  mEditText = (EditText) findViewById(R.id.myEdit);
  mImageView = (ImageView) findViewById(R.id.myImage);
  mButton2.setEnabled(false);
  /* 预览图片的 Button */
  mButton1.setOnClickListener(new Button.OnClickListener()
```

```
    {
      @Override
      public void onClick(View v)
      {
        String path=mEditText.getText().toString();
        if(path.equals(""))
        {
          showDialog("网址不可为空白!");
        }
        else
        {
          /* 传入 type=1 为预览图片 */
          setImage(path,1);
        }
      }
    });
```

② 单击 mButton2 按钮时通过 mButton2.setOnClickListener 将图片设置为桌面。如果网址为空则输出空白提示，如果不为空则传入“type=2”将其设置为桌面。具体代码如下：

```
    /* 将图片设为桌面的 Button */
    mButton2.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        try
        {
          String path=mEditText.getText().toString();
          if(path.equals(""))
          {
            showDialog("网址不可为空白!");
          }
          else
          {
            /* 传入 type=2 为设置桌面 */
            setImage(path,2);
          }
        }
        catch (Exception e)
        {
          showDialog("读取错误!网址可能不是图片或网址错误!");
          bm = null;
          mImageView.setImageBitmap(bm);
          mButton2.setEnabled(false);
          e.printStackTrace();
        }
      }
    });
  }
```

③ 定义方法 setImage(String path,int type)将图片抓取预览并设置为桌面，如果有异常则输出对应提示。具体代码如下：

```
/* 将图片抓下来预览并设置为桌面的方法 */
private void setImage(String path,int type)
{
  try
  {
    URL url = new URL(path);
    URLConnection conn = url.openConnection();
    conn.connect();
    if(type==1)
    {
      /* 预览图片 */
      bm = BitmapFactory.decodeStream(conn.getInputStream());
      mImageView.setImageBitmap(bm);
      mButton2.setEnabled(true);
    }
    else if(type==2)
    {
      /* 设置为桌面 */
      Pingmu.this.setWallpaper(conn.getInputStream());
      bm = null;
      mImageView.setImageBitmap(bm);
      mButton2.setEnabled(false);
      showDialog("桌面背景设置完成!");
    }
  }
  catch (Exception e)
  {
    showDialog("读取错误!网址可能不是图片或网址错误!");
    bm = null;
    mImageView.setImageBitmap(bm);
    mButton2.setEnabled(false);
    e.printStackTrace();
  }
}
```

④ 定义方法 showDialog(String mess)来弹出一个对话框，单击后完成背景设置。具体代码如下：

```
/* 弹出 Dialog 的方法 */
private void showDialog(String mess){
  new AlertDialog.Builder(example8.this).setTitle("Message")
  .setMessage(mess)
  .setNegativeButton("确定", new DialogInterface.OnClickListener()
  {
    public void onClick(DialogInterface dialog, int which)
    {
    }
  })
```

```
    .show();
  }
}
```

(3) 在文件 droidManifest.xml 中需要声明 SET_WALLPAPER 权限和 INTERNET 权限，主要代码如下：

```
<uses-permission android:name="android.permission.SET WALLPAPER"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

执行后在屏幕中显示一个输入框和两个按钮，如图 4-5 所示，输入图片网址并单击“预览”按钮后，可以查看此图片。单击“设置”按钮后可以将此图片设置为屏幕背景。



图 4-5 执行效果

4.3 使用 HttpURLConnection 类

在 java.net 类中，HttpURLConnection 类是一种访问 HTTP 资源的方式。HttpURLConnection 类具有完全的访问能力，可以取代 HttpGet 和 HttpPost 类。本节将详细讲解 HttpURLConnection 的基本用法。

4.3.1 HttpURLConnection的主要用法

在实际项目应用中，使用类 HttpURLConnection 可以实现以下 4 个功能。

1. 从Internet获取网页

在实现此功能时，需要先发送请求，然后将网页以流的形式读回来。

(1) 创建一个 URL 对象：

```
URL url = new URL("http://www.sohu.com");
```

(2) 利用 HttpURLConnection 对象从网络中获取网页数据：

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

(3) 设置连接超时：

```
conn.setConnectTimeout(6* 1000);
```

(4) 对响应码进行判断：

```
if (conn.getResponseCode() != 200) throw new RuntimeException("请求url失败");
```

(5) 得到网络返回的输入流：

```
InputStream is = conn.getInputStream();
String result = readData(is, "GBK");
conn.disconnect();
```

在实现此功能时，必须要记得设置连接超时，如果网络不好，Android 系统在超过默认时间后会收回资源中断操作。如果返回的响应码是 200，则标明成功。利用 ByteArrayOutputStream 类可以将得到的输入流写入内存。由此可见，在 Android 中对文件流的操作和 Java SE 中是一样的。

2. 从Internet获取文件

利用 HttpURLConnection 对象从网络中获取文件数据的基本流程如下。

(1) 创建 URL 对象后传入文件路径：

```
URL url = new URL("http://photocdn.sohu.com/20100125/Img269812337.jpg");
```

(2) 创建 HttpURLConnection 对象后从网络中获取文件数据：

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

(3) 设置连接超时：

```
conn.setConnectTimeout(6* 1000);
```

(4) 对响应码进行判断：

```
if (conn.getResponseCode() != 200) throw new RuntimeException("请求url失
败");
```

(5) 得到网络返回的输入流：

```
InputStream is = conn.getInputStream();
```

(6) 写出得到的文件流：

```
outStream.write(buffer, 0, len);
```

在实现此功能时，当对大文件进行操作时需要将文件写到 SDCard 上面，而不要直接写到手机内存上。并且在操作大文件时，要一边从网络上读，一边往 SDCard 上面写，这样可以减少对手机内存的使用。并且完成功能时，不要忘记及时关闭连接流。

3. 向Internet发送请求参数

利用 HttpURLConnection 对象向 Internet 发送请求参数的基本流程如下。

(1) 将地址和参数存储到 byte 数组中：

```
byte[] data = params.toString().getBytes();
```

(2) 创建 URL 对象：

```
URL realUrl = new URL(requestUrl);
```

(3) 用 HttpURLConnection 对象向网络地址发送请求：

```
HttpURLConnection conn = (HttpURLConnection) realUrl.openConnection();
```

(4) 设置允许输出：

```
conn.setDoOutput(true);
```

(5) 设置不使用缓存：

```
conn.setUseCaches(false);
```

(6) 设置使用 Post 方式发送：

```
conn.setRequestMethod("POST");
```

(7) 设置维持长连接：

```
conn.setRequestProperty("Connection", "Keep-Alive");
```

(8) 设置文件字符集：

```
conn.setRequestProperty("Charset", "UTF-8");
```

(9) 设置文件长度：

```
conn.setRequestProperty("Content-Length", String.valueOf(data.length));
```

(10) 设置文件类型：

```
conn.setRequestProperty("Content-Type","application/x-www-form-
urlencoded");
```

(11) 最后以流的方式输出。

在实现此功能时，发送 Post 请求时必须设置允许输出。建议不要使用缓存，避免出现不应该出现的问题。在开始就用 HttpURLConnection 对象的 setRequestProperty()设置，即生成 HTML 文件头。

4. 向Internet发送XML数据

XML 格式是通信的标准语言，Android 系统也可以通过发送 XML 文件传输数据。实现此功能的基本流程如下。

(1) 将生成的 XML 文件写入到 byte 数组中，并设置为 UTF-8。

```
byte[] xmlbyte = xml.toString().getBytes("UTF-8");
```

(2) 创建 URL 对象并指定地址和参数：

```
URL url = new
URL("http://localhost:8080/itcast/contanctmanage.do?method=readxml");
```

(3) 获得连接：

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

(4) 设置连接超时：

```
conn.setConnectTimeout(6* 1000);
```

(5) 设置允许输出：

```
conn.setDoOutput(true);
```

(6) 设置不使用缓存：

```
conn.setUseCaches(false);
```

(7) 设置以 Post 方式传输：

```
conn.setRequestMethod("POST");
```

(8) 维持长连接：

```
conn.setRequestProperty("Connection", "Keep-Alive");
```

(9) 设置字符集：

```
conn.setRequestProperty("Charset", "UTF-8");
```

(10) 设置文件的总长度：

```
conn.setRequestProperty("Content-Length",
String.valueOf(xmlbyte.length));
```

(11) 设置文件类型：

```
conn.setRequestProperty("Content-Type", "text/xml; charset=UTF-8");
```

(12) 以文件流的方式发送 XML 数据：

```
outStream.write(xmlbyte);
```

注意： 使用 Android 中的 HttpURLConnection 时，有个地方需要引起注意，就是如果你的程序中有跳转，并且跳转有外部域名的跳转，那么非常容易超时并抛出域名无法解析的异常(Host Unresolved)，建议做跳转处理的时候不要使用它自带的方法设置成为自动跟随跳转，最好自己做处理，以便防止莫名其妙的异常。这个问题在模拟器上面看不出来，只有在真机上面才能看出来。

4.3.2 在Android中使用HttpURLConnection类

在编程时我们无须担心普通 Java 平台和 Android 平台的差异，我们完全可以根据 4.3.1 节中的知识在 Android 中使用 HttpURLConnection 类。

1. 在手机屏幕中显示网络中的指定图片

在日常应用中，我们经常不需要将网络中的图片下载在到我们的手机上，只是在网络上浏览一下。此时我们可以使用 HttpURLConnection 类打开链接，这样就可以获取连接数据了。

实　例	功　能	源码路径
实例 4-5	在手机屏幕中显示网络中的图片	下载路径:\daima\4\tu

在本实例中，使用 HttpURLConnection 类连接并获取网络中的数据，将获取的数据用 InputStream(输入流)的方式保存在记忆空间中。本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下：

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:background="@drawable/white"
  android:orientation="vertical"
  android:layout width="fill parent"
  android:layout height="fill parent"
  >
  <TextView
    android:id="@+id/myTextView1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/app name"/>
  <Button
    android:id="@+id/myButton1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="@string/str button1" />
  <ImageView
    android:id="@+id/myImageView1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:layout gravity="center" />
</LinearLayout>
```

(2) 编写主程序文件 tu.java。首先通过方法 getURLBitmap()将图片作为参数传入到创建的 URL 对象，然后通过方法 getInputStream()获取连接图的 InputStream。文件 tu.java 的主要实现代码如下：

```
public class tu extends Activity
{
  private Button mButton1;
  private TextView mTextView1;
  private ImageView mImageView1;
  String uriPic = "http://www.baidu.com/img/baidu sylogo1.gif";
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mButton1 = (Button) findViewById(R.id.myButton1);
    mTextView1 = (TextView) findViewById(R.id.myTextView1);
    mImageView1 = (ImageView) findViewById(R.id.myImageView1);

    mButton1.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        /* 设置 Bitmap 在 ImageView 中 */
```




```
        mImageView1.setImageBitmap(getURLBitmap());
        mTextView1.setText("");
      }
    });
  }

  public Bitmap getURLBitmap()
  {
    URL imageUrl = null;
    Bitmap bitmap = null;
    try
    {
      /* new URL 对象将网址传入 */
      imageUrl = new URL(uriPic);
    }
    catch (MalformedURLException e)
    {
      e.printStackTrace();
    }
    try
    {
      /* 取得连接 */
      HttpURLConnection conn = (HttpURLConnection) imageUrl
          .openConnection();
      conn.connect();
      /* 取得返回的 InputStream */
      InputStream is = conn.getInputStream();
      /* 将 InputStream 变成 Bitmap */
      bitmap = BitmapFactory.decodeStream(is);
      /* 关闭 InputStream */
      is.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
    return bitmap;
  }
}
```

执行后单击“单击后获取网络上的图片”按钮后可以显示指定网址的图片，如图 4-6 所示。



图 4-6　执行效果

2. 在手机屏幕中显示网页

在日常应用中，我们可以使用 HttpURLConnection 类获取某一个网页的内容。

实　例	功　能	源码路径
实例 4-6	在手机屏幕中显示一个网页	下载路径:\daima\4\GetHtml

在本实例中，当我们在文本框中输入网址并单击“显示网页”按钮后，会获取文本框中的网址，打开 HttpURLConnection 连接并获取输入流，然后将返回的流保存为 HTML 文件，最后用 WebView 将 HTML 文件显示出来。本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
   <EditText
    android:id="@+id/myEdit1"
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:maxLines="2"
    android:hint="请输入网址"
   />
   <Button
    android:id="@+id/myButton1"
    android:layout width="100px"
    android:layout height="wrap content"
    android:text="显示网页"
   />
    <WebView
        android:id="@+id/myWeb1"
        android:layout width="fill parent"
        android:layout height="wrap content"
        android:minHeight="200px"
    />
</LinearLayout>
```

(2) 编写主程序文件 GetHtml.java，在方法 getStaticPageByBytes()中通过 HttpURLConnection 来获取某一个网页的内容。文件 GetHtml.java 的具体实现代码如下：

```
package ckl.gethtml;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.EditText;

public class GetHtml extends Activity {
    private EditText mEdit = null;
    private Button mButton = null;
    private WebView mWeb = null;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mEdit = (EditText)findViewById(R.id.myEdit1);
        mButton = (Button)findViewById(R.id.myButton1);
        mWeb = (WebView)findViewById(R.id.myWeb1);

        mWeb.getSettings().setJavaScriptEnabled(true);
        mWeb.getSettings().setPluginsEnabled(true);

        mButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                String strUrl = mEdit.getText().toString();
                String strFile = "/sdcard/test.html";
                if (!strUrl.startsWith("http://")) {
                    strUrl = "http://" + strUrl;
                }
                getStaticPageByBytes(strUrl, strFile);
                mWeb.loadUrl("file://" + strFile);
            }
        });
    }

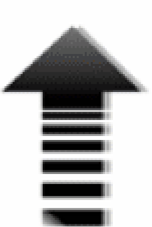
    private void getStaticPageByBytes(String surl, String strFile){

        Log.i("getStaticPageByBytes", surl + ", " + strFile);

        HttpURLConnection connection = null;
        InputStream is = null;

        File file = new File(strFile);
        FileOutputStream fos = null;

        try {
            URL url = new URL(surl);
```



```
            connection = (HttpURLConnection)url.openConnection();

            int code = connection.getResponseCode();
            if (HttpURLConnection.HTTP_OK == code) {
                connection.connect();
                is = connection.getInputStream();
                fos = new FileOutputStream(file);

                int i;
                while((i = is.read()) != -1){
                    fos.write(i);
                }

                is.close();
                fos.close();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (connection != null) {
                connection.disconnect();
            }
        }
    }
}
```

执行后的效果如图 4-7 所示。



图 4-7　执行效果

注意： 本实例的做法并不能每次都保证可以获得正确的网页内容，主要是因为受到网页内容编码(Encoding)的影响，页面编码内容的编码是不确定的，可能不是 UTF-8。因为在 Java 内部是使用 UTF-16 来表示字符的，所以在使用 String 保存页面内容时，会被转换为 UTF-16 来保存，写入文件时再转换为操作系统中的默认编码，这样导致保存文件内容的编码和 html 中指定的编码不一致，导致中文乱码。

第5章

为Android开发网页

Android 手机系统功能十分强大，开发人员可以在上面开发出功能强大的应用程序。在众多网络应用中，网页项目将成为现阶段一个新兴的热点，所以很有必要专门开发能在 Android 手机上浏览的网页。其实本书前面所讲解的 HTML、CSS、JavaScript 都是网页开发技术，用这三种技术开发的网页能够在手机屏幕上正常浏览吗？答案是肯定的，但是前提是需要进行一些变动。本章将详细讲解为 Android 开发 HTML 网页的方法。



5.1 准 备 工 作

开发人员都很希望用 HTML、CSS 和 JavaScript 技术来构建适应于 Android 系统的应用程序。这个旅程的第一步是为 HTML 添加有亲和力的样式，使它们更像移动应用程序。在实现这个功能的时候，我们将 CSS 样式应用到传统的 HTML 网页上，让它们在 Android 手机上正常浏览，并且很容易浏览。

5.1.1 搭建开发环境

在开发能够在 Android 手机运行的网页工作之前，首先需要有一个 Web 空间。这个空间的作用是，将我们开发的网页上传到这个空间，然后就可以在 Android 模拟器中浏览该网页了，这样就实现了测试网页的目的。可能有的读者有自己的 Web 空间，如果没有也不要紧张，可以免费申请一个空间。很多网站提供了免费空间服务，如 http://www.3v.cm/。申请免费空间的基本流程如下。

(1) 登录 http://www.3v.cm/，如图 5-1 所示。



图 5-1 登录http://www.3v.cm/

(2) 单击图 5-1 左侧的“注册”按钮，进入“服务条款”页面，如图 5-2 所示。

(3) 单击“我同意”按钮，注入填写用户名页面，如图 5-3 所示。

(4) 填写完用户名后单击“下一步”按钮，进入填写注册信息页面，如图 5-4 所示。

(5) 填写完注册信息后单击“递交”按钮完成注册。在注册中心页面我们可以管理自己的空间，如图 5-5 所示。



第一步：　请认真阅读服务条款

欢迎您申请使用三维免费个人主页空间服务，为维护网上公共秩序和社会稳定，请您自觉遵守以下条款：

一、不得利用本站危害国家安全、泄露国家秘密，不得侵犯国家社会集体的和公民的合法权益，不得利用本站制作、复制和传播下列信息：

（一）煽动抗拒、破坏宪法和法律、行政法规实施的；
（二）煽动颠覆国家政权，推翻社会主义制度的；
（三）煽动分裂国家、破坏国家统一的；
（四）煽动民族仇恨、民族歧视，破坏民族团结的；
（五）捏造或者歪曲事实，散布谣言，扰乱社会秩序的；
（六）宣扬封建迷信、淫秽、色情、赌博、暴力、凶杀、恐怖、教唆犯罪的；
（七）公然侮辱他人或者捏造事实诽谤他人的，或者进行其他恶意攻击的；
（八）损害国家机关信誉的；
（九）其他违反宪法和法律行政法规的；

二、互相尊重，对自己的言论和行为负责。

我同意　不同意

图 5-2　服务条款页面

第二步：　选择用户名、空间类型

用户名：　*　检测用户名　(请使用3-12个英文字母或数字或-，可任意组合)

空间类型：免费空间-100M　*　→按此查看所选空间详细参数

下一步

图 5-3　填写用户名页面

帐号信息

用户名：qqdsadsad
密码安全性：
密　码：　*（6-12位，区分大小写）
确认密码：　*
密码问题：　*（如：你最爱的人是谁）
问题答案：　*（答案：Nana）

基本资料

真实姓名：　*
性　别：请选择　*
出生日期：　*务必使用IE浏览器，否则有可能无法正常显示
QQ号码：
电子邮箱：　*（用来接收您的帐号信息及密码 ）

网站信息

网站名称：　*
网站分类：请选择　*
网站介绍：在此输入您的网站简介　(建议60字以内)>
验证码：　7908（验证码，看不清楚？请点击刷新）

递交　重填

图 5-4　填写注册信息页面



图 5-5 用户中心页面

(6) 单击图 5-5 左侧的“FTP 管理”链接可以更改我们的 FTP 密码，并且可以查看我们空间的 IP 地址，如图 5-6 所示。



图 5-6 FTP管理

根据图 5-6 中的资料我们可以用专业上传工具上传我们编写的程序文件。

(7) 单击图 5-5 左侧的“文件管理”链接，在弹出的页面中可以在线管理我们空间中的文件，如图 5-7 所示。

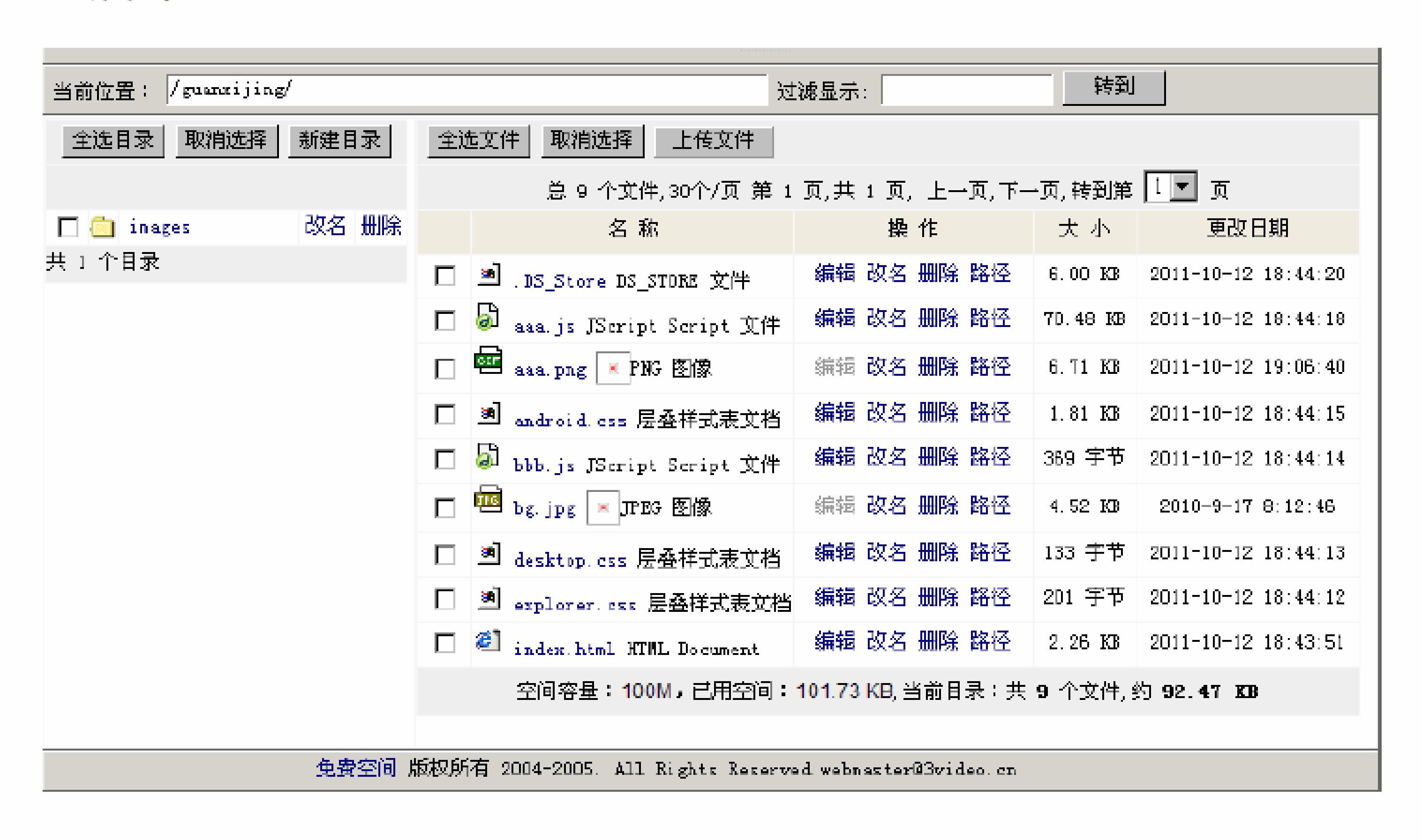
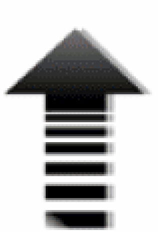


图 5-7 文件管理



单击图 5-7 中每一个文件的“路径”链接，可以获取该文件的 URL 地址，这样我们在 Android 手机中就可以用这个 URL 来访问此文件，查看此文件在 Android 手机中的执行效果。

5.1.2　简单网页开发

下面我们以一个具体例子来作为开始。本实例使用 HTML 和 CSS 技术实现了 4 个简单的网页。

实　例	功　能	源码路径
实例 5-1	编写一个适用于 Android 系统的网页	下载路径:\daima\5\first\

主页文件 index.html 的源代码如下：

```
<html>
   <head>
      <title>aaa</title>
      <link rel="stylesheet" href="desktop.css" type="text/css" />
   <body>
      <div id="container">
         <div id="header">
            <h1><a href="./">AAAA</a></h1>
            <div id="utility">
               <ul>
                  <li><a href="about.html">关于我们</a></li>
                  <li><a href="blog.html">博客</a></li>
                  <li><a href="contact.html">联系我们</a></li>
               </ul>
            </div>
            <div id="nav">
               <ul>
                  <li><a href="bbb.html">Android 之家</a></li>
                  <li><a href="ccc.html">电话支持</a></li>
                  <li><a href="ddd.html">在线客服</a></li>
                  <li><a href="http://www.aaa.com">在线视频</a></li>
               </ul>
            </div>
         </div>
         <div id="content">
            <h2>About</h2>
            <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为
               的，希望事实如此....</p>
         </div>
         <div id="sidebar">
            <img alt="好图片" src="aaa.png">
            <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为
               的，希望事实如此....</p>
         </div>
         <div id="footer">
```

```
            <ul>
                <li><a href="bbb.html">Services</a></li>
                <li><a href="ccc.html">About</a></li>
                <li><a href="ddd.html">Blog</a></li>
            </ul>
            <p class="subtle">巅峰卓越</p>
        </div>
    </div>
</body>
</html>
```

根据“样式和表现想分离”的原则，我们需要单独写一个 CSS 文件，通过这个 CSS 文件来给上述这个网页进行修饰，修饰的最终目的是能够在 Android 手机上进行浏览。

注意：在实际开发应用中，最好将桌面浏览器的样式表和 Android 样式表划清界限。根据笔者的经验，写两个完全独立的文件会舒服很多。当然还有另一种做法是把所有的 CSS 规则放到一个单一的样式表中，但是这种做法不值得提倡，具体原因有以下两点。

- 文件太长了就显得麻烦，不利于维护。
- 把太多不相关的桌面样式规则发送到手机上，将会浪费一些宝贵的带宽和存储空间。

开始写 CSS 文件。为了适应 Android 系统，编写下面的 link 标签。

```
<link rel="stylesheet" type="text/css"
 href="android.css" media="only screen and (max-width: 480px)" />
<link rel="stylesheet" type="text/css"
 href="desktop.css" media="screen and (min-width: 481px)" />
```

在上述代码中，最明显的变动是浏览器宽度的变化，即：

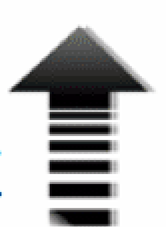
```
max-width: 480px
min-width: 481px
```

这是因为手机屏幕的宽度和电脑屏幕的宽度是不一样的(当然长度也不一样，但是都具有下拉功能)，480 是 Android 系统的标准宽度，上述代码的功能是不管浏览器的窗口是多大，桌面用户看到的都是文件 desktop.css 中样式修饰的页面，宽度都是用如下代码设置的宽度。

```
max-width: 480px
min-width: 481px
```

上述代码中有以下两个 CSS 文件。

- desktop.css 文件：该文件是在开发电脑页面时编写的样式文件，就是为这个 HTML 页面服务的。
- android.css 文件：该文件是一个新文件，也是本章将要讲解的重点，通过该文件，可以将上面的电脑网页显示在 Android 手机中。当读者开发出完整的 android.css 后，可以直接在 HTML 文件中将如下代码删除，就不再用这个修饰文件了。



```
<link rel="stylesheet" type="text/css"
 href="desktop.css" media="screen and (min-width: 481px)" />
```

此时在 Chrome 浏览器浏览修改后的 HTML 文件，无论从 Android 手机浏览器还是电脑浏览器，执行后都将得到一个完整的页面展示。完整代码如下：

```
<html>
   <head>
      <title>AAAA</title>
      <link rel="stylesheet" type="text/css" href="android.css"
         media="only screen and (max-width: 480px)" />
      <link rel="stylesheet" type="text/css" href="desktop.css"
         media="screen and (min-width: 481px)" />
      <!--[if IE]>
         <link rel="stylesheet" type="text/css" href="explorer.css"
            media="all" />
      <![endif]-->
      <script type="text/javascript" src="jquery.js"></script>
      <script type="text/javascript" src="android.js"></script>
   <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
   </head>
   <body>
      <div id="container">
        <div id="header">
            <h1><a href="./">AAAA</a></h1>
            <div id="utility">
               <ul>
                  <li><a href="about.html">关于我们</a></li>
                  <li><a href="blog.html">博客</a></li>
                  <li><a href="contact.html">联系我们</a></li>
               </ul>
            </div>
            <div id="nav">
               <ul>
                  <li><a href="bbb.html">Android 之家</a></li>
                  <li><a href="ccc.html">电话支持</a></li>
                  <li><a href="ddd.html">在线客服</a></li>
                  <li><a href="http://www.aaa.com">在线视频</a></li>
               </ul>
            </div>
         </div>
         <div id="content">
            <h2>About</h2>
            <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为
               的，希望事实如此....</p>
         </div>
         <div id="sidebar">
            <img alt="好图片" src="aaa.png">
            <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为
               的，希望事实如此....</p>
         </div>
```

```
            <div id="footer">
                <ul>
                    <li><a href="bbb.html">Services</a></li>
                    <li><a href="ccc.html">About</a></li>
                    <li><a href="ddd.html">Blog</a></li>
                </ul>
                <p class="subtle">巅峰卓越</p>
            </div>
        </div>
    </body>
</html>
</html>
```

desktop.css 文件的代码如下：

```
For example:
body {
    margin:0;
    padding:0;
    font: 75% "Lucida Grande", "Trebuchet MS", Verdana, sans-serif;
}
```

执行效果如图 5-8 所示。

AAAA

- 关于我们
- 博客
- 联系我们

- Android之家
- 电话支持
- 在线客服
- 在线视频

About

欢迎大家学习Android，都说这是一个前途辉煌的职业，我也是这么认为的，希望事实如此.... ...



欢迎大家学习Android，都说这是一个前途辉煌的职业，我也是这么认为的，希望事实如此....

- Services
- About
- Blog

巅峰卓越

图 5-8　执行效果

5.1.3　控制页面的缩放

除非我们明确告诉 Android 浏览器，否则它会认为页面宽度是 980px。当然这在大多数情况下能工作得很好，因为电脑已经适应了这个宽度。但是如果针对小尺寸屏幕的 Android 手机的话，我们必须做一些调整，需要在 HTML 文件的 head 元素中加一个 viewport 元标签，让移动浏览器知道屏幕的大小。



```
<meta name="viewport" content="user-scalable=no, width=device-width" />
```

这样就实现了屏幕的自动缩放，可以根据显示屏的大小带给我们不同大小的显示页面。读者无须担心加上 viewport 后在电脑上的显示影响，因为桌面浏览器会忽略 viewport 元标签。

如果不设置 viewport 的宽度，页面在加载后会缩小。我们不知道缩放的大小是多少，因为 Android 浏览器的设置项允许用户设置默认缩放大小，选项有大、中(默认)、小。即使设置过 viewport 宽度，这个设置项也会影响页面的缩放大小。

5.2　为 Android 中的网页添加 CSS 样式

我们接着上一节的演示代码继续讲解。前面代码中的 android.css 文件一直没用，下面将开始编写这个文件，目的是使我们的网页在 Android 手机上完美并优越显示。

5.2.1　编写基本样式

所谓的基本样式是指诸如背景颜色、字体大小、字体颜色等样式，在上一节实例的基础上继续扩展，看下面的具体实现流程。

(1) 在 android.css 文件中设置<body>元素的如下基本样式。

```
body {
    background-color: #ddd;      /* 背景颜色 */
    color: #222;                 /* 字体颜色 */
    font-family: Helvetica;      /* 字体 */
    font-size: 14px;             /* 字体大小 */
    margin: 0;                   /* 外边距 */
    padding: 0;                  /* 内边距 */
}
```

(2) 开始处理<header>中的<div>内容，它包含主要入口的链接(也就是 logo)和一级、二级站点导航。第一步是把 logo 链接的格式调整得像可以点击的标题栏，在此我们将下面的代码加入到 android.css 文件中。

```
#header h1 {
    margin: 0;
    padding: 0;
}
#header h1 a {
    background-color: #ccc;
    border-bottom: 1px solid #666;
    color: #222;
    display: block;
    font-size: 20px;
    font-weight: bold;
    padding: 10px 0;
    text-align: center;
```

```
    text-decoration: none;
}
```

(3) 用同样的方式格式化一级和二级导航的<ul>元素。在此只需用通用的标签选择器(也就是#header ul)就够用了，而不必再设置标签<ID>，也就不必设置诸如下面的样式了。

- #header ul
- #utility
- #header ul
- #nav

实现此过程的代码如下：

```
#header ul {
    list-style: none;
    margin: 10px;
    padding: 0;
}
#header ul li a {
    background-color: #FFFFFF;
    border: 1px solid #999999;
    color: #222222;
    display: block;
    font-size: 17px;
    font-weight: bold;
    margin-bottom: -1px;
    padding: 12px 10px;
    text-decoration: none;
}
```

(4) 为 content 和 sidebar div 加内边距，使文字到屏幕边缘之间有空的距离。代码如下。

```
#content, #sidebar {
    padding: 10px;
}
```

(5) 设置<footer>中内容的样式，<footer>里面的内容比较简单，我们只需将 display 设置为 none，代码如下：

```
#footer {
    display: none;
}
```

此时将上述代码在电脑中执行的效果如图 5-9 所示。



AAAA

- 关于我们
- 博客
- 联系我们

- Android之家
- 电话支持
- 在线客服
- 在线视频

About

欢迎大家学习Android，都说这是一个前途辉煌的职业，我也是这么认为的，希望事实如此....



欢迎大家学习Android，都说这是一个前途辉煌的职业，我也是这么认为的，希望事实如此....

- Services
- About
- Blog

巅峰卓越

图 5-9　电脑中的执行效果

在 Android 模拟器中的执行效果如图 5-10 所示。



图 5-10　在Android中的执行效果

因为添加了自动缩放，并且添加了修饰 Menu 的样式，所以整个界面看上去很美。

5.2.2　添加视觉效果

为了使我们的页面变得精彩，我们可以尝试加一些充满视觉效果的样式。

(1) 给<header>文字加 1px 向下的白色阴影，背景加上 CSS 渐变效果。具体代码如下：

```
#header h1 a {
   text-shadow: 0px 1px 1px #fff;
   background-image: -webkit-gradient(linear, left top, left bottom,
      from(#ccc), to(#999));
}
```

对于上述代码有两点说明。

- text-shadow 声明：参数从左到右分别表示水平偏移、垂直偏移、模糊效果和颜色。在大多数情况下，可以将文字设置成上面代码中的数值，这在 Android 界面中的显示效果也不错。在大部分浏览器上，将模糊范围设置为 0px 也能看到效果。但 Android 要求模糊范围最少是 lpx，如果设置成 0px，在 Android 设备上将显示不出来文字阴影。
- -webkit-gradient：功能是让浏览器在运行时产生一张渐变的图片。因此，可以把 CSS 渐变功能用在任何平常指定图片(比如背景图片或者列表式图片)URL 的地方。参数从左到右的排列顺序分别是：渐变类型(可以是 linear 或者 radial 的)、渐变起点(可以是 left top、left bottom、right top 或者 right bottom)、渐变终点、起点颜色和终点颜色。

注意： 在上述赋值中不能颠倒描述渐变起点、终点常量(left top、left bottom、right top、right bottom)的水平和垂直顺序。也就是说，top left、bottom left、top right 和 bottom right 是不合法的值。

(2) 为导航菜单加上圆角样式，代码如下：

```
#header ul li:first-child a {
   -webkit-border-top-left-radius: 8px;
   -webkit-border-top-right-radius: 8px;
}
#header ul li:last-child a {
   -webkit-border-bottom-left-radius: 8px;
   -webkit-border-bottom-right-radius: 8px;
}
```

上述代码使用-webkit-border-radius 属性描述角的方式，定义列表第一个元素的上两个角和最后一个元素的下两个角是以 8 像素为半径的圆角。此时在 Android 中的执行效果如图 5-11 所示。



图 5-11　在Android中的执行效果

此时会发现列表显示样式变为了圆角样式，整个外观显得更加圆滑和自然。

5.3　为 Android 网页添加 JavaScript 特效

经过前面的步骤，一个基本的 HTML 页面就设计完成了，并且这个页面可以在 Android 手机上完美显示。为了使页面更加完美，接下来的内容中，我们将详细讲解在上述页面中添加 JavaScript 行为特效的基本知识。

5.3.1　jQuery框架介绍

jQuery 是继 prototype 之后又一个优秀的 JavaScript 框架。jQuery 是一个轻量级的 js 库，压缩后只有 21KB。jQuery 不但兼容 CSS 3，而且还兼容各种浏览器。jQuery 不但使用户可以更方便地处理 HTML Documents、Events 和动画元素，并且方便地为网站提供 Ajax 交互。jQuery 还有一个比较大的优势是，它的文档说明很全，而且各种应用也说得很详细，同时还有许多成熟的插件可供选择。jQuery 能够使用户的 HTML 页保持代码和 HTML 内容分离，也就是说，不用再在 HTML 里面插入一堆 js 来调用命令了，只需定义 id 即可。

1. jQuery的语法

jQuery 是为 HTML 元素的选取编制的，通过 jQuery 可以对 HTML 元素执行某些操作。使用 jQuery 的基础语法格式如下：

```
$(selector).action()
```

- 美元符号：定义 jQuery。
- 选择符(selector)：“查询”和“查找” HTML 元素。
- * jQuery 的 action()：执行对元素的操作。

例如下面的代码：

```
$(this).hide()       //隐藏当前元素
$("p").hide()        //隐藏所有段落
$("p.test").hide()   //隐藏所有 class="test" 的段落
$("#test").hide()    //隐藏所有 id="test" 的元素
```

2. jQuery的简单使用

下面通过如下代码来让读者认识 jQuery 的强大功能。

```
<html>
<head>
<script type="text/javascript" src="/jquery/jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
  $("button").click(function(){
    $("#test").hide();
  });
```

```
});
</script>
</head>

<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p id="test">This is another paragraph.</p>
<button type="button">Click me</button>
</body>

</html>
```

上述代码演示了 jQuery 中函数 hide()的基本用法，功能是隐藏当前的 HTML 元素。执行效果如图 5-12 所示，只显示一个按钮。单击该按钮后，会隐藏所有的 HTML 元素，包括这个按钮，此时页面一片空白。

Click me

图 5-12　未被隐藏时

注意：本书的重点不是 jQuery，所以不再对其使用知识进行讲解。读者可以参阅其他书籍或网上教程来学习它。

5.3.2　具体实践

本小节继续以前面的实例 5-1 为基础。下面步骤的目的是为页面添加一些 JavaScript 元素，让页面支持一些基本的动态行为。在具体实现的时候，使用了 jQuery 框架。具体要做的是，让用户控制是否显示页面顶部那个太引人注目的导航栏，这样用户可以只在想看的时候去看。我们的实现流程如下。

(1) 隐藏<header>中的 ul 元素，让它在用户第一次加载页面之后不会显示出来。具体代码如下：

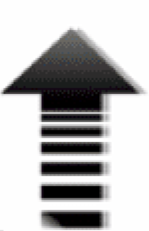
```
#header ul. hide{
display: none;
)
```

(2) 定义显示和隐藏菜单的按钮，代码如下：

```
<div class="leftButton"onclick="toggleMenu()">Menu</div>
```

我们定义一个带有 leftButton 类的 div 元素，将其放在 header 里面。下面是这个按钮的完整 CSS 样式代码：

```
#header div.leftButton {
   position: absolute;
   top: 7px;
   left: 6px;
   height: 30px;
```



```
    font-weight: bold;
    text-align: center;
    color: white;
    text-shadow: rgba (0,0,0,0.6) 0px -1px 1px;
    line-height: 28px;
    border-width: 0 8px 0 8px;
    -webkit-border-image: url(images/button.png) 0 8 0 8;
}
```

上述代码的具体说明如下。

- position: absolute：从顶部开始，设置 position 为 absolute，相当于把这个 div 元素从 HTML 文件流中去掉，从而可以设置自己的最上面和最左面的坐标。
- height: 30px：设置高度为 30px。
- font-weight:bold：定义文字格式为粗体，白色带有一点向下的阴影，在元素里居中显示。
- text-shadow:rgba：rgb(255,255,255)、rgb(100%,100%,100%)格式和#FFFFFF 格式是一个原理，都是设置颜色值的。在 rgba()函数中，它的第 4 个参数用来定义 Alpha 值(透明度)，取值范围从 0～1。其中 0 表示完全透明，1 表示完全不透明，0～1 之间的小数表示不同程度的半透明。
- line-height：把元素中的文字往下移动的距离，使之不会和上边框齐平。
- border-width 和-webkit-border-image：这两个属性一起决定把一张图片的一部分放入某一元素的边框中去。如果元素大小由于文字的增减而改变，图片会自动拉伸以适应这样的变化。这一点其实非常棒，意味着只需要不多的图片、少量的工作、低带宽和更少的加载时间。
- border-width：让浏览器把元素的边框定位在距上 0px、距右 8px、距下 0px、距左 8px 的地方(4 个参数从上开始，以顺时针为序)。不需要指定边框的颜色和样式。边框宽度定义好之后，就要确定放进去的图片了。
- url(images/button.png) 0 8 0 8：5 个参数从左到右分别是：图片的 URL、上边距、右边距、下边距、左边距(再一次，从上顺时针开始)。URL 可以是绝对(比如 http://example.com/myBorderlmage.png)或者相对路径，后者是相对于样式表所在的位置的，而不是引用样式表的 HTML 页面的位置。

(3) 在 HTML 文件中插入引入 JavaScript 的代码，将对 aaa.js 和 bbb.js 的引用写到 HTML 文件中。

```
<script type="text/javascript" src="aaa.js"></script>
<script type="text/javascript" src="bbb.js"></script>
```

在文件 bbb.js 中，我们编写一段 JavaScript 代码，这段代码的主要作用是让用户显示或者隐藏 nav 菜单。代码如下：

```
if (window.innerWidth && window.innerWidth <= 480) {
    $(document).ready(function(){
        $('#header ul').addClass('hide');
        $('#header').append('<div class="leftButton"
onclick="toggleMenu()">Menu</div>');
```

```
    });
    function toggleMenu() {
        $('#header ul').toggleClass('hide');
        $('#header .leftButton').toggleClass('pressed');
    }
}
```

对上述代码的具体说明如下。

第 1 行：括号中的代码，表示当 Window 对象的 innerWidth 属性存在并且 innerWidth 小于等于 480px(这是大部分手机合理的最大宽度值)时才执行到内部。该行保证只有当用户使用 Android 手机或者类似大小的设备访问这个页面时，上述代码才会执行。

第 2 行：使用了函数 document ready，此函数是“网页加载完成”函数。这段代码的功能是设置当网页加载完成之后才运行里面的代码。

第 3 行：使用了典型的 jQuery 代码，目的是选择 header 中的<ul>元素并且往其中添加 hide 类开始。此处的 hide 是前面 CSS 文件中的选择器，这行代码执行的效果是隐藏 header 的 ul 元素。

第 4 行：此处是给 header 添加按钮的地方，目的是让我们可以显示和隐藏菜单。

第 8 行：函数 toggleMenu()用 jQuery 的 toggleClass()函数来添加或删除所选择对象中的某个类。这里应用了 header 的 ul 里的 hide 类。

第 9 行：在 header 的 leftButton 里添加或删除 pressed 类，类 pressed 的具体代码如下：

```
#header div.pressed {
    -webkit-border-image: url(images/button clicked.png) 0 8 0 8;
}
```

通过上述样式和 JavaScript 行为设置以后，Menu 开始动起来了，默认是隐藏了链接内容，单击之后才会在下方显示链接信息，如图 5-13 所示。

图 5-13　下方显示信息



5.4　在 Android 网页中使用 Ajax 特效

Ajax 是指异步 JavaScript 和 XML 技术的称呼，是 Asynchronous JavaScript And XML 的缩写。Ajax 不是一种新的编程语言，而是一种用于创建更好更快以及交互性更强的 Web 应用程序的技术。通过使用 Ajax，我们可使用 JavaScript 的 XMLHttpRequest 对象来直接与服务器进行通信。通过这个对象，我们可在不重载页面的情况下与 Web 服务器交换数据。

Ajax 在浏览器与 Web 服务器之间使用异步数据传输(HTTP 请求)，这样就可使网页从服务器请求少量的信息，而不是整个页面。

既然 Ajax 和 JavaScript 的关系如此密切，那么就很有必要在开发的 Android 网页中使用 Ajax，这样可以给用户带来更精彩的体验。

本节将从一个具体例子开始，讲解 Ajax 在 Android 网页中的简单应用。

实　例	功　能	源码路径
实例 5-2	在 Android 系统中开发一个 Ajax 网页	下载路径:\daima\5\gaoji\

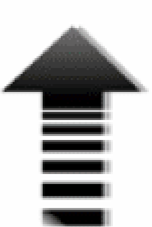
(1) 编写一个名为 android.html 的 HTML 文件，具体代码如下：

```
<html>
   <head>
      <title>Jonathan Stark</title>
      <meta name="viewport" content="user-scalable=no, width=device-
          width" />
      <link rel="stylesheet" href="android.css" type="text/css"
          media="screen" />
      <script type="text/javascript" src="jquery.js"></script>
      <script type="text/javascript" src="android.js"></script>
   </head>
   <body>
      <div id="header"><h1>AAA</h1></div>
      <div id="container"></div>
   </body>
</html>
```

(2) 编写样式文件 android.css，主要代码如下：

```
body {
   background-color: #ddd;
   color: #222;
   font-family: Helvetica;
   font-size: 14px;
   margin: 0;
   padding: 0;
}
#header {
   background-color: #ccc;
```

```
    background-image: -webkit-gradient(linear, left top, left bottom,
        from(#ccc), to(#999));
    border-color: #666;
    border-style: solid;
    border-width: 0 0 1px 0;
}
#header h1 {
    color: #222;
    font-size: 20px;
    font-weight: bold;
    margin: 0 auto;
    padding: 10px 0;
    text-align: center;
    text-shadow: 0px 1px 1px #fff;
    max-width: 160px;
    overflow: hidden;
    white-space: nowrap;
    text-overflow: ellipsis;
}
ul {
    list-style: none;
    margin: 10px;
    padding: 0;
}
ul li a {
    background-color: #FFF;
    border: 1px solid #999;
    color: #222;
    display: block;
    font-size: 17px;
    font-weight: bold;
    margin-bottom: -1px;
    padding: 12px 10px;
    text-decoration: none;
}
ul li:first-child a {
    -webkit-border-top-left-radius: 8px;
    -webkit-border-top-right-radius: 8px;
}
ul li:last-child a {
    -webkit-border-bottom-left-radius: 8px;
    -webkit-border-bottom-right-radius: 8px;
}
ul li a:active, ul li a:hover {
    background-color: blue;
    color: white;
}
#content {
    padding: 10px;
    text-shadow: 0px 1px 1px #fff;
}
```



```
#content a {
    color: blue;
}
```

上述样式文件在本章的前面内容中都进行过详细讲解，相信读者一读便懂。

(3) 继续编写如下 HTML 文件。

- about.html
- blog.html
- contact.html
- consulting-clinic.html
- index.html

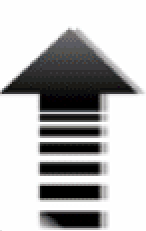
为了简单，它们的代码都是一样的，具体代码如下：

```
<html>
    <head>
        <title>AAA</title>
        <meta name="viewport" content="user-scalable=no, width=device-
            width" />
        <link rel="stylesheet" type="text/css" href="android.css"
            media="only screen and (max-width: 480px)" />
        <link rel="stylesheet" type="text/css" href="desktop.css"
            media="screen and (min-width: 481px)" />
        <!--[if IE]>
            <link rel="stylesheet" type="text/css" href="explorer.css"
                 media="all" />
        <![endif]-->
        <script type="text/javascript" src="jquery.js"></script>
        <script type="text/javascript" src="android.js"></script>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    </head>
    <body>
        <div id="container">
      <div id="header">
                <h1><a href="./">AAAA</a></h1>
                <div id="utility">
                    <ul>
                        <li><a href="about.html">AAA</a></li>
                        <li><a href="blog.html">BBB</a></li>
                        <li><a href="contact.html">CCC</a></li>
                    </ul>
                </div>
                <div id="nav">
                    <ul>
                        <li><a href="bbb.html">DDD</a></li>
                        <li><a href="ccc.html">EEE</a></li>
                        <li><a href="ddd.html">FFF</a></li>
                        <li><a href="http://www.aaa.com">GGG</a></li>
                    </ul>
                </div>
```

```
            </div>
            <div id="content">
                <h2>About</h2>
                <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为
                    的，希望事实如此....</p>
            </div>
            <div id="sidebar">
                <img alt="好图片" src="aaa.png">
                <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为
                    的，希望事实如此....</p>
            </div>
            <div id="footer">
                <ul>
                    <li><a href="bbb.html">Services</a></li>
                    <li><a href="ccc.html">About</a></li>
                    <li><a href="ddd.html">Blog</a></li>
                </ul>
                <p class="subtle">巅峰卓越</p>
            </div>
        </div>
    </body>
</html>
```

(4) 编写 JavaScript 文件 android.js，在此文件中使用了 Ajax 技术，具体代码如下：

```
var hist = [];
var startUrl = 'index.html';
$(document).ready(function(){
    loadPage(startUrl);
});
function loadPage(url) {
    $('body').append('<div id="progress">wait for a moment...</div>');
    scrollTo(0,0);
    if (url == startUrl) {
        var element = ' #header ul';
    } else {
        var element = ' #content';
    }
    $('#container').load(url + element, function(){
        var title = $('h2').html() || '你好!';
        $('h1').html(title);
        $('h2').remove();
        $('.leftButton').remove();
        hist.unshift({'url':url, 'title':title});
        if (hist.length > 1) {
            $('#header').append('<div class=
               "leftButton">'+hist[1].title+'</div>');
            $('#header.leftButton').click(function(e){
                $(e.target).addClass('clicked');
                var thisPage = hist.shift();
                var previousPage = hist.shift();
```



```
            loadPage(previousPage.url);
        });
    }
    $('#container a').click(function(e){
        var url = e.target.href;
        if (url.match(/aaa.com/)) {
            e.preventDefault();
            loadPage(url);
        }
    });
    $('#progress').remove();
  });
}
```

对于上述代码的具体说明如下。

- 第 1～5 行：使用了 iQuery 的 document ready 函数，目的是使浏览器在加载页面完成后运行 loadPage()函数。
- 剩余的行数是函数 loadPage(url)部分，此函数的功能是载入地址为 URL 的网页，但是在载入时使用了 Ajax 技术特效。具体说明如下。
  - 第 7 行：为了使 Ajax 效果能够显示出来，loadPage()函数启动时，在 body 中增加一个正在加载的 div，然后在 hijackLinks()函数结束的时候删除。
  - 第 9～13 行：如果没有在调用函数的时候指定 URL(比如第一次在 document ready 函数中调用)，URL 将会是 undefined，这一行会被执行。这一行和下一行是 iQuery 的 load()函数样例。load()函数在为页面增加简单快速的 Ajax 实用性上非常出色。如果把这一行翻译出来，它的意思是“从 index.html 中找出所有#header 中的 ul 元素，并把它们插入到当前页面的#container 元素中，完成之后再调用 hijackLinks()函数”。当 URL 参数有值的时候，执行第 12 行。从效果上看，“从传给 loadPage()函数的 URL 中得到#content 元素，并把它们插入到当前页面的#container 元素，完成之后调用 hijackLinks()函数。

(5) 最后的修饰。

为了能使我们设计的页面体现出 Ajax 效果，还需继续设置样式文件 android.css。

① 为了能够显示出“加载中…”的样式，需要在 android.css 中添加如下对应的修饰代码：

```
#progress {
    -webkit-border-radius: 10px;
    background-color: rgba(0,0,0,.7);
    color: white;
    font-size: 18px;
    font-weight: bold;
    height: 80px;
    left: 60px;
    line-height: 80px;
    margin: 0 auto;
    position: absolute;
```

```
    text-align: center;
    top: 120px;
    width: 200px;
}
```

② 用边框图片修饰返回按钮，并清除默认点击后高亮显示的效果。在文件 android.css 中添加如下修饰代码：

```
#header div.leftButton {
    font-weight: bold;
    text-align: center;
    line-height: 28px;
    color: white;
    text-shadow: 0px -1px 1px rgba(0,0,0,0.6);
    position: absolute;
    top: 7px;
    left: 6px;
    max-width: 50px;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    border-width: 0 8px 0 14px;
    -webkit-border-image: url(images/back button.png) 0 8 0 14;
    -webkit-tap-highlight-color: rgba(0,0,0,0);
}
```

此时在 Android 中执行上述文件，执行后先加载页面，在加载时会显示“wait for a moment...”的提示，如图 5-14 所示。在滑动选择某个链接的时候，被选中的会有不同的颜色，如图 5-15 所示。

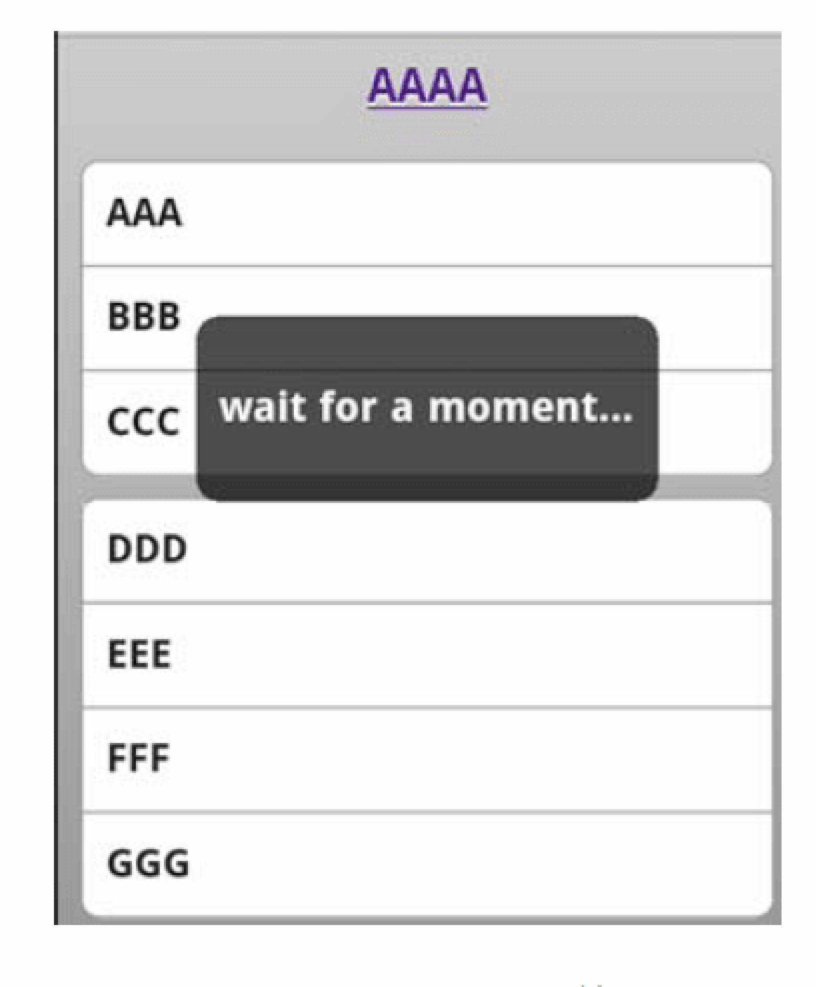


图 5-14　提示特效

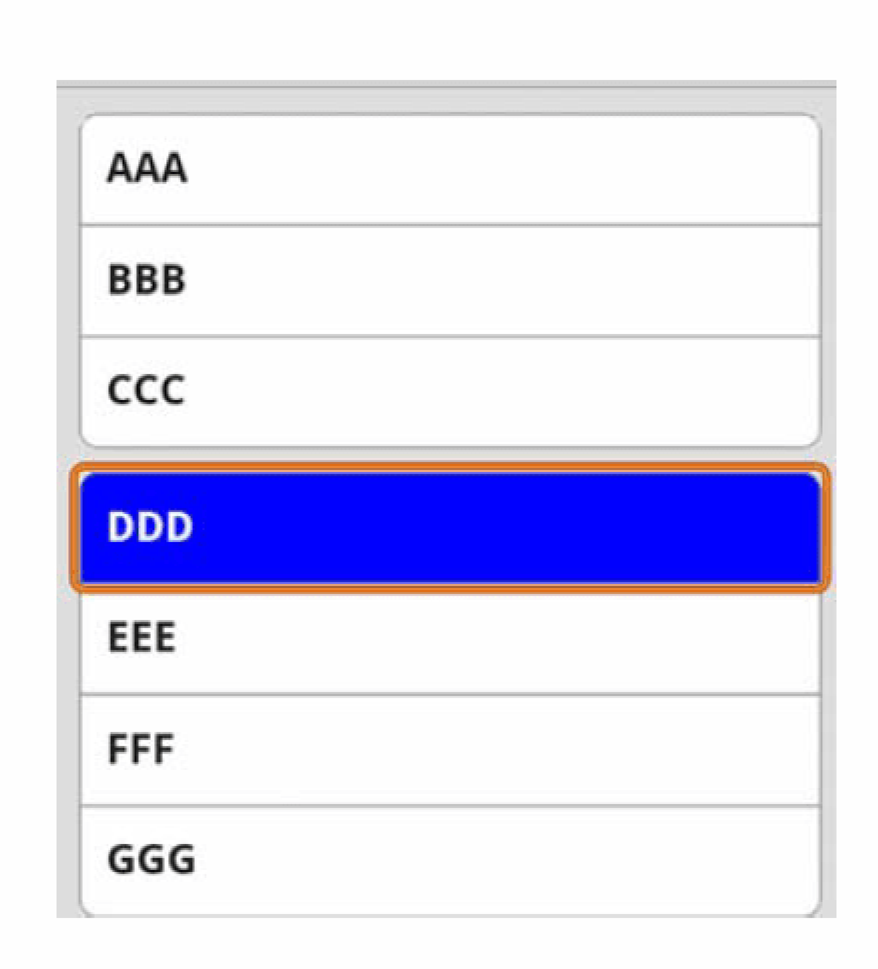


图 5-15　被选择的不同颜色

而文件 android.html 的执行效果和其他文件相比稍有不同，如图 5-16 所示。这是因为在编码时有意而为之的。

hello!

AAA
BBB
CCC
DDD
EEE
FFF
GGG

图 5-16　文件android.html的执行效果

5.5　让 Android 网页充满灵动活力

除了可以为 Android 网页实现 Ajax 特效之外，还可以为其添加动画效果，这样会让我们的 Android 网页更加充满活力。

5.5.1　开源框架——JQTouch

JQTouch 为我们提供了一系列功能的 jQuery 插件，这些功能可以为手机浏览器 WebKit 服务。目前，随着 Android 手机、iPhone、iTouch、iPad 等产品的流行，越来越多的开发者想开发相关的应用程序。到目前，苹果只提供了 Objective-C 语言去编写 iPhone 应用程序。但可惜的是，即使苹果的总裁乔布斯吹嘘它的易用性，C 语言本身是不容易学习的语言，跟开发 Web 网站来比却更加复杂。不过这一切将发生变化，因为 jQuery 的工具 JQTouch 出现了。

使用 JQTouch 的目的是使构建基于 Android 和 iPhone 的应用变得更加容易，而所有的只需要一点 HTML、CSS 和一些 JavaScript 知识，就能够创建可在 WebKit 浏览器上(iPhone、Android、Palm Pre)运行的手机应用程序。

读者可以在其官方网址 http://www.jqtouch.com/下载资源。因为是开源的，所以下载后可以直接使用。

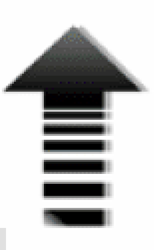
5.5.2　JQTouch简单应用

接下来将以一个具体的实例，来详细讲解使用 JQTouch 框架为 Android 实现动画效果的过程。

实　例	目　的	源码路径
实例 5-3	在 Android 系统中使用 JQTouch 框架开发网页	下载路径:\daima\5\donghua\

首先编写一个名为 index.html 的 HTML 文件，具体代码如下：

```
<!DOCTYPE html>
<html>
    <head>
        <title>AAA</title>
        <link type="text/css" rel="stylesheet" media="screen"
            href="jqtouch/jqtouch.css">
        <link type="text/css" rel="stylesheet" media="screen"
            href="themes/jqt/theme.css">
        <script type="text/javascript" src="jqtouch/jquery.js"></script>
        <script type="text/javascript" src="jqtouch/jqtouch.js"></script>
        <script type="text/javascript">
            var jQT = $.jQTouch({
                icon: 'kilo.png'
            });
        </script>
    </head>
    <body>
        <div id="home">
            <div class="toolbar">
                <h1>Data</h1>
                <a class="button flip" href="#settings">Settings</a>
            </div>
            <ul class="edgetoedge">
                <li class="arrow"><a href="#dates">Dates</a></li>
                <li class="arrow"><a href="#about">About</a></li>
            </ul>
        </div>
        <div id="about">
            <div class="toolbar">
                <h1>About</h1>
                <a class="button back" href="#">Back</a>
            </div>
            <div>
                <p>Choose you food.</p>
            </div>
        </div>
        <div id="dates">
            <div class="toolbar">
                <h1>Time</h1>
                <a class="button back" href="#">Back</a>
            </div>
            <ul class="edgetoedge">
                <li class="arrow"><a id="0" href="#date">AAA</a></li>
                <li class="arrow"><a id="1" href="#date">BBB</a></li>
                <li class="arrow"><a id="2" href="#date">CCC</a></li>
                <li class="arrow"><a id="3" href="#date">DDD</a></li>
                <li class="arrow"><a id="4" href="#date">EEE</a></li>
                <li class="arrow"><a id="5" href="#date">FFF</a></li>
            </ul>
        </div>
```



```
        <div id="date">
            <div class="toolbar">
                <h1>Time</h1>
                <a class="button back" href="#">Back</a>
                <a class="button slideup" href="#createEntry">+</a>
            </div>
            <ul class="edgetoedge">
                <li id="entryTemplate" class="entry" style="display:none">
                    <span class="label">Label</span> <span class=
                     "calories">000</span> <span class="delete">Delete</span>
                </li>
            </ul>
        </div>
        <div id="createEntry">
            <div class="toolbar">
                <h1>WHY</h1>
                <a class="button cancel" href="#">Cancel</a>
            </div>
            <form method="post">
                <ul class="rounded">
                    <li><input type="text" placeholder="Food" name="food" id=
                      "food" autocapitalize="off" autocorrect="off"
                      autocomplete="off" /></li>
                    <li><input type="text" placeholder="Calories" name=
                      "calories" id="calories" autocapitalize="off"
                      autocorrect="off" autocomplete="off" /></li>
                    <li><input type="submit" class="submit" name="waction"
                      value="Save Entry" /></li>
                </ul>
            </form>
        </div>
        <div id="settings">
            <div class="toolbar">
                <h1>Control</h1>
                <a class="button cancel" href="#">Cancel</a>
            </div>
            <form method="post">
                <ul class="rounded">
                    <li><input placeholder="Age" type="text" name="age"
                      id="age" /></li>
                    <li><input placeholder="Weight" type="text" name=
                      "weight" id="weight" /></li>
                    <li><input placeholder="Budget" type="text" name=
                      "budget" id="budget" /></li>
                    <li><input type="submit" class="submit" name="waction"
                      value="Save Changes" /></li>
                </ul>
            </form>
        </div>
    </body>
</html>
```

接下来开始对上述代码进行详细讲解。

(1) 通过如下代码启用 JQTouch 和 jQuery：

```
<script type="text/javascript" src="jqtouch/jquery.js"></script>
<script type="text/javascript" src="jqtouch/jqtouch.js"></script>
```

(2) 实现 home 面板，具体代码如下：

```
<div id="home">
   <div class="toolbar">
      <h1>Data</h1>
      <a class="button flip" href="#settings">Settings</a>
   </div>
   <ul class="edgetoedge">
      <li class="arrow"><a href="#dates">Dates</a></li>
      <li class="arrow"><a href="#about">About</a></li>
   </ul>
</div>
```

对应的效果如图 5-17 所示。

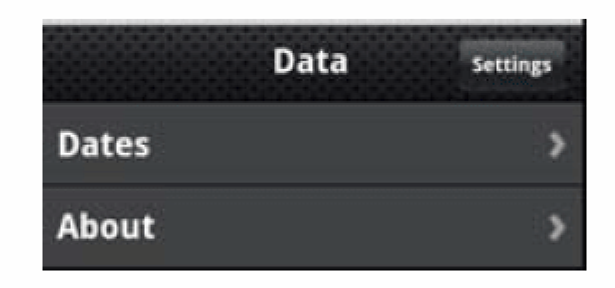


图 5-17　home面板

(3) 实现 about 面板，具体代码如下。

```
<div id="about">
   <div class="toolbar">
      <h1>About</h1>
      <a class="button back" href="#">Back</a>
   </div>
   <div>
      <p>Choose you food.</p>
   </div>
</div>
```

对应的效果如图 5-18 所示。

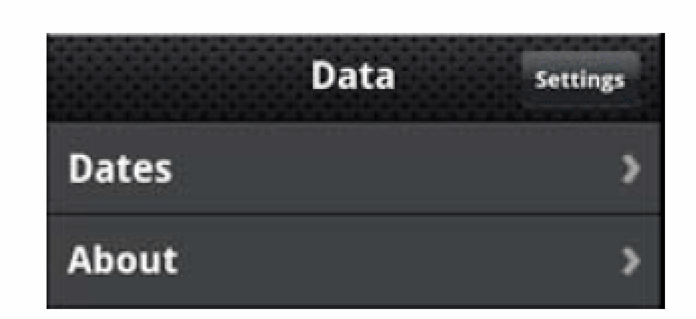


图 5-18　about面板

(4) 实现 dates 面板，具体代码如下：

```
<div id="dates">
   <div class="toolbar">
      <h1>Time</h1>
```

```
            <a class="button back" href="#">Back</a>
        </div>
        <ul class="edgetoedge">
            <li class="arrow"><a id="0" href="#date">AAA</a></li>
            <li class="arrow"><a id="1" href="#date">BBB</a></li>
            <li class="arrow"><a id="2" href="#date">CCC</a></li>
            <li class="arrow"><a id="3" href="#date">DDD</a></li>
            <li class="arrow"><a id="4" href="#date">EEE</a></li>
            <li class="arrow"><a id="5" href="#date">FFF</a></li>
        </ul>
    </div>
```

对应的效果如图 5-19 所示。



图 5-19　dates面板

(5) 实现 date 面板，具体代码如下：

```
    <div id="date">
        <div class="toolbar">
            <h1>Time</h1>
            <a class="button back" href="#">Back</a>
            <a class="button slideup" href="#createEntry">+</a>
        </div>
        <ul class="edgetoedge">
            <li id="entryTemplate" class="entry" style="display:none">
                <span class="label">Label</span> <span class=
                  "calories">000</span> <span class="delete">Delete</span>
            </li>
        </ul>
    </div>
```

(6) 实现 settings 面板，具体代码如下：

```
    <div id="settings">
        <div class="toolbar">
            <h1>Control</h1>
            <a class="button cancel" href="#">Cancel</a>
        </div>
        <form method="post">
            <ul class="rounded">
                <li><input placeholder="Age" type="text" name="age"
```

```
id="age" /></li>
                <li><input placeholder="Weight" type="text" name=
                  "weight" id="weight" /></li>
                <li><input placeholder="Budget" type="text" name=
                  "budget" id="budget" /></li>
                <li><input type="submit" class="submit" name="waction"
                  value="Save Changes" /></li>
            </ul>
        </form>
    </div>
```

对应的效果如图 5-20 所示。

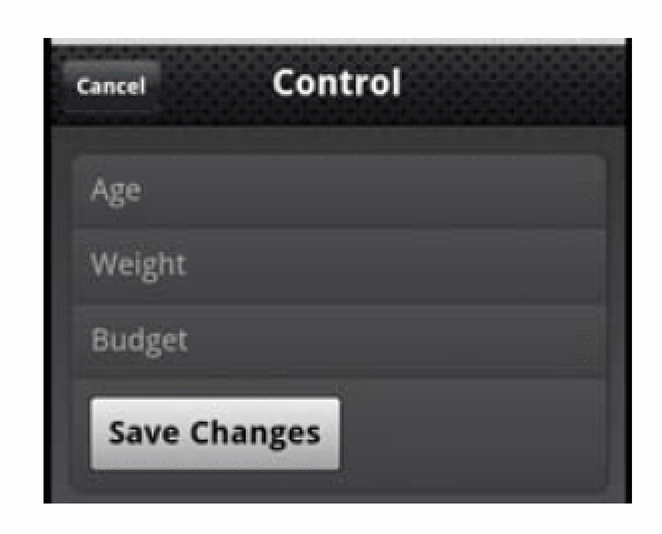
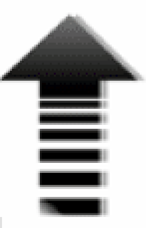


图 5-20　settings面板

接下来看样式文件 theme.css。此样式文件非常简单，功能是对 index.html 中的元素进行修饰。其实图 5-17～图 5-20 都是经过 theme.css 修饰之后的显示效果。主要代码如下：

```
body {
    background: #000;
    color: #ddd;
}
#jqt > * {
    background: -webkit-gradient(linear, 0% 0%, 0% 100%, from(#333),
        to(#5e5e65));
}
#jqt h1, #jqt h2 {
    font: bold 18px "Helvetica Neue", Helvetica;
    text-shadow: rgba(255,255,255,.2) 0 1px 1px;
    color: #000;
    margin: 10px 20px 5px;
}
/* @group Toolbar */
#jqt .toolbar {
    -webkit-box-sizing: border-box;
    border-bottom: 1px solid #000;
    padding: 10px;
    height: 45px;
    background: url(img/toolbar.png) #000000 repeat-x;
    position: relative;
}
#jqt .black-translucent .toolbar {
    margin-top: 20px;
}
```



```
#jqt .toolbar > h1 {
   position: absolute;
   overflow: hidden;
   left: 50%;
   top: 10px;
   line-height: 1em;
   margin: 1px 0 0 -75px;
   height: 40px;
   font-size: 20px;
   width: 150px;
   font-weight: bold;
   text-shadow: rgba(0,0,0,1) 0 -1px 1px;
   text-align: center;
   text-overflow: ellipsis;
   white-space: nowrap;
   color: #fff;
}
#jqt.landscape .toolbar > h1 {
   margin-left: -125px;
   width: 250px;
}
#jqt .button, #jqt .back, #jqt .cancel, #jqt .add {
   position: absolute;
   overflow: hidden;
   top: 8px;
   right: 10px;
   margin: 0;
   border-width: 0 5px;
   padding: 0 3px;
   width: auto;
   height: 30px;
   line-height: 30px;
   font-family: inherit;
   font-size: 12px;
   font-weight: bold;
   color: #fff;
   text-shadow: rgba(0, 0, 0, 0.5) 0px -1px 0;
   text-overflow: ellipsis;
   text-decoration: none;
   white-space: nowrap;
   background: none;
   -webkit-border-image: url(img/button.png) 0 5 0 5;
}
#jqt .button.active, #jqt .cancel.active, #jqt .add.active {
   -webkit-border-image: url(img/button clicked.png) 0 5 0 5;
   color: #aaa;
}
#jqt .blueButton {
   -webkit-border-image: url(img/blueButton.png) 0 5 0 5;
   border-width: 0 5px;
}
```

```
#jqt .back {
    left: 6px;
    right: auto;
    padding: 0;
    max-width: 55px;
    border-width: 0 8px 0 14px;
    -webkit-border-image: url(img/back button.png) 0 8 0 14;
}
#jqt .back.active {
    -webkit-border-image: url(img/back button clicked.png) 0 8 0 14;
}
#jqt .leftButton, #jqt .cancel {
    left: 6px;
    right: auto;
}
#jqt .add {
    font-size: 24px;
    line-height: 24px;
    font-weight: bold;
}
#jqt .whiteButton,
#jqt .grayButton, #jqt .redButton, #jqt .blueButton, #jqt .greenButton {
    display: block;
    border-width: 0 12px;
    padding: 10px;
    text-align: center;
    font-size: 20px;
    font-weight: bold;
    text-decoration: inherit;
    color: inherit;
}

#jqt .whiteButton.active, #jqt .grayButton.active,
#jqt .redButton.active, #jqt .blueButton.active,
#jqt .greenButton.active,
#jqt .whiteButton:active, #jqt .grayButton:active,
#jqt .redButton:active, #jqt .blueButton:active,
#jqt .greenButton:active {
    -webkit-border-image: url(img/activeButton.png) 0 12 0 12;
}
#jqt .whiteButton {
    -webkit-border-image: url(img/whiteButton.png) 0 12 0 12;
    text-shadow: rgba(255, 255, 255, 0.7) 0 1px 0;
}
#jqt .grayButton {
    -webkit-border-image: url(img/grayButton.png) 0 12 0 12;
    color: #FFFFFF;
}
```

上述代码只是 theme.css 的 1/5，具体内容请读者参考本书下载资源中的源码。因为里面的内容都在本书前面的知识中讲解过，所以在此不再占用篇幅。

到此为止，我们的页面就能够动起来了，每一个页面的切换都具有了动画效果，如图 5-21 所示。



图 5-21　闪烁的动画效果

书的截图体现不出动画效果，建议读者在模拟器上亲自实践体验。

注意： 其实最后的步骤就是使用 JQTouch 了，因为是开源部分，所以无须笔者耗费篇幅，笔者做的工作只是设置了里面的几个属性而已。文件 jqtouch.js 比较长，读者想理解 JQTouch 开源代码的各个部分，可以参阅相关资料。如果个人 JavaScript、Ajax、CSS、HTML 水平很不错，建议下载开源代码自己分析。网上也有很多参考资料，现在比较著名的是 LUPA 社区中的在线分析教程。URL 地址如下：

http://code.lupaworld.com/code.php?mod=list&itemid=39&path=./kissy_1.1.7/

此教程界面清新，左侧是导航，十分便于我们的浏览，如图 5-22 所示。



图 5-22　JQTouch在线源码分析

5.6 为网页增加数据存储功能

大多数软件程序都需要某种持久化的方式来存储有用的数据。对于网络应用程序，这个任务一般会交给服务器端的数据库或者在浏览器的 cookie 中来完成。伴随着 HTML 5 的出现，Web 开发者有了另外两种选择：Web Storage 和 Web SQL Database。

5.6.1 在Android网页中使用Web Storage

Web Storage 有两种形式，分别是 localStorage(本地存储)和 sessionStorage(会话存储)。这两种形式都允许开发者使用 JavaScript 设置键值对，并在重新加载不同页面的时候读出它们，这一点和现在 cookie 机制非常类似。但是和 cookie 不同的是，Web Storage 的数据是完全存储在客户端的，无须通过浏览器的请求再传输到服务器。由此可见，和 cookie 方式相比，Web Storage 可以在本地存储更多的数据。

localStorage 和 sessionStorage 在功能上是一样的，只是在持久性和范围上有所不同。

- localStorage：即使已经关闭了浏览窗口，数据也会被保存起来并可用于所有来自同源(必须有相同的域名、协议和端口)窗口(或者标签页)的加载，这对参数设置或是偏好设置之类的功能非常有用。
- sessionStorage：数据存储在窗口对象中，对于其他窗口或者标签页不可见，并且当窗口关闭时会丢失数据。sessionStorage 可用于特殊的窗口状态，例如一个标签页的高亮状态或者一个表格的排序状态。例如在下面的代码中，sessionStorage 都可以用 localStorage 来代替，但是在窗口或者标签页关闭时，会丢失使用 sessionStorage 存储的数据。

设置参数的方法非常简单，例如：

```
localStorage.setItem('age'.40);
```

访问一个存储的数据的方法非常简单，例如：

```
var age=localStorage.getItem('age');
```

可以这样删除一个特定的键值对：

```
localStorage. removeItem('age');
```

或者删除所有的键值对：

```
localStorage.clear();
```

假设我们的键是有效的 JavaScript token，比如没有空格、除下划线之外的标点，则可以使用如下语法：

```
    localStorage.age=4e            //设置 age 的值
    var age=localStorage.age;      //取得 age 的值
    delete localStorage.age;       //从存储中删除 age
```



其实 localStorage 和 sessionStorage 中键的存储是分开的。如果你在两种存储中使用同样的键名也是没有冲突的。

1. 将用户设置保存到localStorage

下面介绍一个实际的例子，目的是更新本章前面例子中的 settings 面板，此时可以让它把表单数据存储到 localStorage 当中。接下来会有相当数量的 JavaScript 代码，但是并不准备把它们都放进 HTML 文件的 head 标签中。为了保持代码的工整，在存放 HTML 文档的相同路径下创建一个名为 123.js 的文件，同时更新 HTML 文件的 head 部分来引用 123.js。

```
<head>
<title>123</title>
<link type="text/css" rel="stylesheet" media="screen"
href="jqtouch/jqtouch.css">
<link type="text/css" rel="stylesheet" media="screen"
href="themes/jqt/theme.css">
<script type="text/javasc ript" src="jqtouch/jquery.js"></script>
<script type="text/javasc ript" src="jqtouch/jqtouch.js"></script>
<script type="text/javascript" src="123.js"</script>
</head>
```

由此可见，在更新的同时也从 HTML 的 head 部分删除了 jQTouch 的构造函数。其实它并没有被删除，只是被移到文件 123.js 文件中罢了。但需要确保从 HTML 的 head 部分中删除上述提到的内容并且在相同路径下建立如下内容的 123.js 文件，然后在浏览器中刷新主 HTML 文件以确保它仍旧可以工作：

```
var jQT=$.jQTouch({
icon:'123.png'
});
```

代码经过重新组织以后，可以添加保存设置的代码了。需要重写 settings 表单的提交动作的代码，并用一个名为 saveSettings()的自定义函数替换。因为用到了 jQTouch，所以只需修改 Document Ready 函数中的一行代码即可。将下面的代码加入到 123.js 文件中：

```
$(document).ready(function(){
$('#settings form') .submit(saveSettings);
```

这段代码的作用是，当用户提交 settings 表单时，用 saveSettings()函数的运行代替表单的提交动作。当调用 saveSettings()函数时，会用 jQTouch 的 val()函数获得表单的 3 个输入项，并且分别存入 localStorage 这个同名变量中。将下面的函数加入 123.js 文件中：

```
function saveSettings()  {
localStorage.age=$('#age').val();
localStorage.budget=$('#budget').val();
localStorage.weight=$('#weight').val.();
jQT.goBack();
return false;
}
```

一旦数值被保存，便调用 jQTouch 的 goBack()函数(从第 2 行到最后一行)来关闭面板并返回前一个页面。接下来，返回 false 给触发这个函数的提交事件，以防进行默认的提交操作。如果没有这行代码，现有页面会被重新加载一次，这显然不是我们希望的。

到此为止已经可以运行程序，前往 settings 面板输入设置，然后提交表单将设置存储到 localStorage 中。由于在提交表单时没有清空相关字段，当用户再次访问 settings 面板时，上次输入的数据仍然会存在。然而这不是因为使用了 localStorage 保存的缘故，而是因为只要输入过后不清理，这些字段会一直在那里。因此，当用户下次重新运行程序并访问 settings 面板时，即使这些字段被保存了，它们仍会丢失。

为了解决这个问题，需要使用函数 loadSettings()来加载配置，所以在 123.js 文件中添加以下函数：

```
function loadSettings()  {
$('#age').val(localStorage.age);
$('#budget').val(localStorage.budget);
$('#weight').val(localStorage.weight);
)
```

函数 loadSettings()和函数 saveSettings()是相反的，它使用 localStorage 中的相应字段来调用 jQTouch 的函数 val()来设置 settings 表单中的 3 个字段值。

接下来需要用方法来触发已有的 loadSettings()函数，触发时刻是当程序启动时。为了实现这个功能，在 123.js 的 document ready 函数中添加如下代码：

```
$(document).ready(function(){
$('#settings form').submit(saveSettings);
loadSettings();
})
```

但是遗憾的是，在程序启动加载配置时有一个漏洞：如果当用户访问 settings 面板，改动了一些值，然后点击 Cancel 按钮而非提交表单，下次程序就不能加载正确的配置。在这种情况下，当用户再次访问 settings 面板时，会显示最新的修改。这不是因为这些新的数据被保存了(实际上并没有)，而只是界面显示的问题。如果用户关闭并重新启动程序，显示的数据又会变成之前保存的数据，因为 loadSettings()函数会在程序启动的时候还原字段中的数据。

其实有很多方法可以改变这种情况，但最合适的方法应该是：每次当 settings 面板移动时，无论进入还是退出，都需要刷新显示的数据。在 jQTouch 中提供了一种简单的方法将 loadSettings()函数和 settings 面板的 pageAnimationStart 事件绑定起来。将刚刚加入的代码用如下加粗代码来代替：

```
$(document).ready(function(){
$('#settings form').submit(saveSettings);
$('#settings') .bind('pageAnimationStart',loadSettings);
});
```

现在 123.js 中的 JavaScript 代码为 settings 面板提供了持久化的数据存储。当再看实现此功能的代码时，发现其实并不算多。以下是到目前为止 123.js 中的代码：




```
var jQT=$.jQTouch({
icon: '123.png'
>);
$(document).ready(function(){
$('#settings form').submit(saveSettings);
$('#settings').bind('pageAnimationStart',loadSettings);
});
function loadSettings()  {
$('#age').val(localStorage.age);
$('#budget').val(localStorage.budget);
$('#weight').val(localStorage.weight);
)
function saveSettings()  {
localStorage.age=$('#age').val();
localStorage.budget=$('#budget').val();
localStorage.weight=$('#weight').val();
jQT.goBack();
return false:
}
```

2. 将选中的数据保存到sessionStorage中

我们最终的目的是配置 date 面板，使得每次显示时它都会根据输入的日期检索数据库中所有的记录，然后在边对边(Edge-To-Edge)的列表中显示出来。这就要求 date 面板知道用户在 dates 面板上点击了哪个日期。

另外还希望允许用户在数据库中添加或者删除条目，所以需要支持 date 面板上已经存在的“+”按钮和 Delete 按钮。第一步是当用户从 dates 面板访问 date 面板时，需要让 date 面板知道用户点击的是哪个条目。有了这个信息，就可以计算出合适的日期的上下文语境。为了实现它，需要在 123.js 的 document reday 函数中添加如下代码：

```
$(document).ready(function(){
$('#settings form').submit(saveSettings);
$('#settings').bind('pageAnimationStart',loadSettings);
$('#dates li a').click(function(){                //(1)
var dayOffset=this.id;                            //(2)
var date=new Date();                              //(3)
date.setDate(date.getDate() - dayOffset);
sessionStorage.currentDate=date.getHonth()+1+'/'+
date.getDate()+'/'+
date.getFullYear();                               //(4)
refreshEntries();                                 //(5)
});
});
```

(1) JQTouch 中的 click()函数将它随后的 JavaScript 代码和 dates 面板中 click 事件的链接绑定起来。

(2) 获取被点击对象的 ID，并且将其存入 dayOffset 变量中。dates 面板中的链接都有范围从 0～5 的 ID，所以被点击的链接的 ID 就相当于点击日期所需要计算的天数，比如，过去 0 天表示今天，过去 1 天表示昨天，过去 2 天表示前天……

(3) 创建了一个新的 JavaScript 日期对象，并且将其存入一个叫 date 的变量中。首先，这个 date 会被设置成创建的日期，所以在下一行中，会从 getDate()函数的结果中减去 dayOffset，再用 setDate()函数将日期设置成选中的日期。其中 dayOffset 为 0 表示今天，1 表示昨天，依此类推。

(4) 生成了一个“MM/DD/YYYY”格式的日期字符串，并将其作为当前日期(currentDate)存入 sessionStorage。

(5) 调用 refreshEntries()函数。此函数的作用是，基于用户点击 dates 面板的日期值来正确更新 date 面板。现在只要使用用户选中的日期来更新 dates 面板工具栏上的标题，就能看到它起作用了。如果没有这个函数，运行后只会出现“Date”字样，而不会出现数据。函数 refreshEntries()应添加到文件 123.js 中：

```
function refreshEntries()  {
    var currentDate=sessionStorage.currentDate;
    $('#date  hl').text (currentDate);
  )
```

5.6.2　在Android网页中使用Web SQL Database

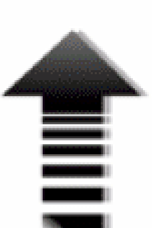
在所有 HTML 5 的特性中，Web SQL Database 的功能非常强大。Web SQL Database 提供给开发者一个简单而强大的 JavaScript 数据库 API，使得可以在一个本地 SQLite 数据库中持久保存数据。其实从技术上讲，Web SQL Database 并非 HTML 5 的一部分，它其实是摆脱了 HTML 5 的细则，并将其融入自己的细则当中。但一般提到它，还是会说是 HTML 5 的特性。

开发者可以使用标准 SQL 语句来创建数据表及插入、更新、查找和删除行。JavaScript Database API 甚至能够支持 transaction(事务)。SQL 有与生俱来的复杂性，但无论如何，这是一个改变游戏规则的变化，所以关注它本身可能会更有意义。

1. 创建数据库

现在用户已经选择好 date 面板的所需数据了，我们已经掌握所有让用户创建条目所必要的信息。在编写 createEntry()函数之前，还要建立一个数据库来存储提交的数据(这是一个一次性的操作)。可以在 123.js 中添加如下代码来实现这项功能：

```
    var db;                                          //(1)
    $(dacument).ready(function(){
    $('#settings form') .submit (saveSettings);
    $('#settings').bind('pageAnimationSta rt',loadSettings);
    $('#dates li a').click(functian(){
    var dayOffset=this.id;
    var date=new Date();
    date.setDate(date.getDate() - dayOffset);
    sessionStorage.currentDate=date.getMonth()+1+'/'+
    date. getDate()+'/'+
    date. getFullYear();
    refreshEntries();
    });
```



```
var shortName='123';                                        //(2)
var version='1.0';
var displayName='123';
var maxSize=65536;
db=openDatabase(shortName,version,displayName,maxSize);     //(3)
db.transaction(                                             //(4)
function(transaction){                                      //(5)
transaction.executeSql(                                     //(6)
'CREATE TABLE IF NOT EXISTS entries '+
'  (id INTEGER NOT NULL PRIMARY KEY AUTOINCREHENT, '+
'   date DATE NOT NULL, food TEXT NOT NULL, '+
'calories INTEGER NOT NULL); '
);
}
);
});
```

(1) 首先要注意的是，在程序里有一个名为 db 的全局变量。一旦建立一个数据库连接，这个变量会保存连接的引用。之所以被设置成全局变量，是因为我们将会在程序的各个地方都用到它。

(2) 接下来的 4 行代码定义了调用 openDatabase 需要的参数。

- shortName：一个指向硬盘上的数据库文件的字符串。
- version：当需要修改数据库模式时用来管理升级和向后兼容的数字。例如，每次程序启动时检查 version 字段，如果过期，创建一个新的数据库并且将旧数据库中的数据移到新数据库中。
- displayName：对用户显示的字符串。例如，displayName 会显示在 Chrome 主界面上的 Developer Tools 里的 Storage 标签中(View→Developer→Developer Tools)。
- maxSize：允许数据库增长到的最大 KB 数。

(3) 当参数被设置以后，这行代码将会调用 openDatabase 并且将数据库连接存储到 db 变量中。如果数据库不存在，将会新建一个。

(4) 所有的数据库查询都必须放在一个事务的上下文当中，所以这里调用了一个 db 对象的 transaction 方法。而接下来的几行代码会构造一个函数作为唯一的参数传入 transaction 方法。

(5) 从本行开始是一个匿名函数，将 transaction 对象当作参数传入。

(6) 一旦进入这个函数，调用 transaction 对象的 executeSql 方法来执行一个标准的 CREATE TABLE 查询语句。其中的 IF NOT EXISTS 子句避免在数据表已经存在的情况下再次创建数据表。

如果再次启动程序，会在 Android 手机上创建一个叫作 123 的数据库。在 Chrome 的桌面版中，实际上是可以浏览并且操作客户端的数据库的，只需要导航到 View→Developer→Developer Tools，单击 Storage 选项。在 Chrome 桌面版当中的 Developer Tools 对于调试来说相当有用。默认情况下，这个工具栏会作为浏览器的一个面板出现。如果你点击浮动图标(鼠标在左下角的图标上悬停一下看它们的作用都是什么)，这个面板会作为

一个独立的窗口显示。通过点击数据库名，这个界面甚至允许执行任意的 SQL 查询语句。

2. 插入行

现在我们有了一个配置好的数据库来接收数据条目，可以生成一个 createEntry()函数了。首先，要重写#createEntry 表单的提交事件，可以通过将 createEntry()函数和 123.js 中 document ready 函数的提交事件绑定到一起来达成此目的。下面只显示了前几行，详情参见粗体代码：

```
$(document).ready(function(){
$('#createEntry form').submit(createEntry);
$('#settings form').submit(saveSettings);
$('#settings').bind('pageAnimationStart',loadSettings);
```

这样，每次用户提交#createEntry 表单，createEntry()函数就会被调用。然后在 123.js 中添加如下代码来在数据库中创建记录：

```
function createEntry()  {
var date=sessionStorage.currentDate;            //(1)
var calories=$('#calories') .val();
var food=$('#food').val();
db.transaction(                                 //(2)
function(transaction){
transaction.executeSql(
'INSERT INTO entries (date, calories, food) VALUES(?,?,?);',
[date, calories, food],
function(){
refreshEntries();
jQT. goBack();
},
errorHandler
);
}
);
return false;
)
```

(1) 这部分包含了我们将会在 SQL 查询语句中使用的一些变量。前面讲到“将选中的数据保存到 sessionStorage 中”，用户在 dates 面板上点击的日期数据已经保存在 sessionStorage.currentDate 中。使用之前需在 settings 表单中用相同的方法从表单中获得另外两个值(calories 和 food)。

(2) 这段代码打开一个数据库事务，并且执行 executeSql()调用。这时我们传入 executeSql()方法的 4 个参数。

- 'INSERT INTO entries (date, calories, food) VALUES(?,?,?);'：这是即将被执行的语句。问号代表数据占位符。
- [date,calories, food]：这个数值数组将被传入数据库当中，它们和 SQL 语句中的问号占位符一一对应。
- function() {refreshEntries();jQT.goBack();}：如果 SQL 查询语句成功返回，这个匿



名函数将会被调用。

- errorHandler：如果 SQL 语句执行失败，这是将会被执行的错误处理函数。

假设插入数据正确，作为第 3 个参数的匿名函数将会被执行。这个函数会调用 refreshEntries()函数。现在这个函数只能更新 date 面板的标题，但是过一会儿就会看到创建的条目显示在列表当中，并且模拟在 Cancel 按钮上的点击，关闭 New Entry 面板回到 date 面板。和在 settings 面板中看到的一样，Cancel 按钮不会取消提交动作，它只是一个标着“Cancel”的返回按钮，不过长得不像一个左箭头。

如果插入不成功，errorHandler()函数会被执行。在文件 123.js 中添加如下代码：

```
function errorHandler(transaction, error){
alert('Oops. Error was'+error.message+'  (Code'+error.code+')'):
return true;
)
```

这个错误处理函数会传入两个参数：transaction 对象和 error 对象。这里使用 error 对象来提示用户错误信息和抛出的错误码。错误处理函数必须返回 true 或 false。如果一个错误处理函数返回 true，比如这是一个致命的错误，执行过程会被停止并且整个事务将会回滚，如果错误处理函数返回 false，比如这不是一个致命的错误，则执行过程会继续进行。

在有些情况下，可能需要根据不同的错误类型判断返回 true 或 false。其实除了 error 对象以外，errorHandler()函数还接收了一个 transaction 对象。可以想象在一些情况下，开发人员需要在错误处理函数中执行 SQL 语句。比如，将错误记录到日志，为调试记录一些元数据或者记录崩溃报告等。可见这个 transaction 对象允许更多的来自错误处理函数内部的 executeSql()调用。这只是一个例子，如果语句当中提到的 errors 表没有建立的话，这句将不会执行：

```
function errorHandler(transaction, error){
alert('Oops. Error was'+error.message+'  (Code'+error.code+-).);
t ransaction.executeSql('INSERT INTO errors (code, message)
  VALUES(?,?);',
[error.code,error.message]);
return false;
}
```

有一点要特别注意：如果希望执行 executeSql()函数中的语句，那么错误处理函数必须返回 false。如果返回的是 true 或者没有返回值，整个事务(包括错误处理当中的这条 SQL 语句)都将会回滚，这样就达不到希望的效果了。

3. 检索行及处理结果集

下面需要扩展 refreshEntries()函数来完成更多功能，而不只是将选择的日期数据更新到标签栏上。具体来讲，下面将会查询数据库中被选择的日期条目，并使用 HTML 中的隐藏 entryTemplate 把它们添加到#date ul 元素中。在这里把 date 面板中的代码再次展示出来，因为它们已经在文件 index.html 中了，所以无须添加。

```
<div id="date">
<div class="toolbar">
<hl>Date</hl>
```

```
<a class="button back" href="#">Back</a>
<a class="button slideup"href="#createEntry">+</a>
</div>
<ul class="edgetoedge">
<li id="ent ryTemplate"class="entry"style="display:none">
<span class="label">Label</span>
<span class="calories">000</span>
<span class="delete">Delete</span>
</li>
</ul>
</div>
```

注意上面的加粗代码：li 元素的样式属性已经设置成 display:none，这会使该元素不在页面上显示出来。这样做是因为使用了 HTML 代码段作为数据库行显示的模板。

下面是完整的 refreshEntries()函数，需要将文件 123.js 中现有的 refreshEntries()的代码替换成如下代码：

```
function refreshEntries()  {
var currentDate=sessionStorage.currentDate;                  //(1)
$('#date hl').text(currentDate);
$('#date ul ti:gt(0) ').remove();                            //(2)
db.transaction(                                              //(3)
function(transaction)  {
transaction.executeSql(
'SELECT*FROM entries WHERE date=7 ORDER BY food;',               //(4)
[currentDate],                                               //(5)
function (transaction.result){                               //(6)
for (var i=0;i<result.rows.length; i++){
var row=result.rows.item(i);                                 //(7)
var newEntryRow=$('#entryTemplate').clone();                 //(8)
newEntryRow. removeAttr('id');
newEntryRaw.removeAttr('style');
newEntryRow.data('entryId'.row.id);                          //(9)
newEntryRow.appendTo('#date uI');                            //(10)
newEntryRow.find('*label').text(row.food);
newEntryRow.find('.calories').text(row.calories);
}
),
errorHandler
);
)
);
}
```

(1) 这两行代码用 sessionStorage 中保存的 currentDate 值来设置 date 面板中的标题栏。

(2) 这一行使用 jQTouch 的 gt()函数(gt 代表“greater than”)来查找并移除任何索引大于 0 的 li 元素。尽管这个函数第一次不会起作用，但因为只有一个 ID 为 entryTemplate 的 li 元素，它的索引为 0 并且是隐藏的，当后续访问该页面时，就需要将数据库中的数据添加到 li 元素之前，将原有的数据删除。否则数据记录会在列表中出现多次，因为相同的数



据将会一遍又一遍地添加进列表。

(3) 这 3 行设置数据库事务和 executeSql 函数。

(4) 本行包含了 executeSql 函数的第一个参数。它是一个简单的 SELECT 语句，问号代表一个数据占位符。

(5) 这是只有一个元素的数组，包含了现有被检索到的数据。在 SQL 查询语句中，问号将会被该元素代替。

(6) 当正确的查询完成时这个匿名函数将被调用。它接收两个参数：transaction 和 result。成功处理函数中的 transaction 对象将向数据库发送新的查询请求，这和之前的错误处理函数一样。然而，在当下不必那样，所以这里不会用 transaction 对象。

此处的 result 对象有 3 个只读属性。

- rowsAffected：用来判断插入、更新和删除查询过后被影响的行数。
- linsertId：返回在插入操作中最后一条被插入行的主键值。
- rows：代表被查询到的记录。rows 对象包含 0 个或者多个 row 对象，并且包含一个 length 属性，在下面的 for 循环代码中可以看到。

(7) 本行使用 rows 的 item()方法把查询到的行的内容设置到 row 变量中。

(8) 这行代码中，使用 clone()方法来复制模板 li，并且在下两行中删除掉它的 id 和 style 属性。删除掉 style 属性会使这行可见，而删除掉 id 是很重要的，否则在页面的结尾就会出现多个相同 id 的数据项。

(9) 本行中将 row 的 id 属性作为数据存入 li 元素中，需要这个数据以便在删除的时候知道这是哪条记录。

(10) 这行代码将 li 元素加入其父元素 ul 当中。接下来的两行用 row 中相应的数据更新 li 的子元素 label 和 calories。

当这些都做完以后，date 面板会对从数据库中检索出来的数据，每一行显示一个 li 元素。每一行都有一个 lable、calories 和 Delete 按钮。如果再创建一些行，就需要加上 CSS，让界面变得更美观一些。所以将下面的 CSS 代码保存到 123.css 文件中，注意将这个文件放在和 HTML 同一层目录下。

```
    #date ut li{
    position:relative;
    )
    #date ul li span{
    color: #FFFFFF;
    text-shadow:0 lpx 2px rgba(0,0,0,.7);
    )
#date ul li.delete {
   position: absolute;
   top: 5px;
   right: 6px;
   font-size: 12px;
  line-height: 30px;
   padding: 0 3px;
   border-width: 0 5px;
   -webkit-barder-image: url(themes/jqt/img/button.png) 0 5 0 5;
}
```

然后在文件 index.html 的 head 部分添加如下代码以便连接到 123.css。

```
<link type="text/css"rel="stylesheet"media="sc reen"href="123.css">
```

4. 删除行

虽然现在 Delete 按钮看起来更像一个按钮了，但单击这些按钮时，它们没有任何反应。这是因为它们是使用 span 标签来实现的按钮，在 HTML 页面中是一个无法交互的元素。

为了让 Delete 按钮起作用，需要使用 jQuery 将它们和点击事件绑定起来。在之前处理 date 面板中的条目时做过类似的事情，用的就是 jQuery 的 click()函数。可惜的是，这种方法在这里行不通。与 dates 面板中的条目不同，date 面板中的条目不是静态的，这说明它们的添加和删除和用户的 session 相关。事实上，当程序启动时，date 面板上并没有可见的条目，因此也没有什么可以绑定到 click 事件上。

解决方案是当 Delete 按钮在 refreshEntries()函数被创建时，将它们绑定到 click 事件上。为了做到这点，需要在 for 循环的最后添加如下加粗的代码：

```
newEnt ryRow.find('.calories').text( row.calories);
newEntryRow.find('.delete').click(function(){        //(1)
var clickedEntry=$(this).parent();                   //(2)
var clickedEntryId=clickedEntry.data('entryId');     //(3)
deleteEntryById(clickedEntryId);                     //(4)
clickedEntry.slideUp();
});
)
```

(1) 这个函数指在 date 元素的 ID 中寻找任何有 delete 这个类的元素，并设置它们的 click()函数。这个 click()函数允许用一个匿名函数作为参数来处理点击事件。

(2) 当 click 事件被触发以后，Delete 按钮的父元素(这里是 li)被找到，然后被保存进 clickedEntry 变量。

(3) 这行代码使用 refreshEntries()函数创建的 li 元素保存的 entryId 值来设置 clickedEntryId 变量。

(4) 这行代码将被点击元素的 ID 传入 deleteEntryById()函数中，接下来的一行调用 jQuery 的 slideUp()函数很好地将 li 元素从界面上移除。

在文件 123.js 中添加 deleteEntryById()函数，使得从数据库中移除一条记录。

```
function deleteEntryById(id)  {
db.transaction(
function(transaction){
transaction.executeSql('DELETE FROM entries WHERE id=?;'
[id],  null, errorHandler);
}
);
)
```

依照前一个例子中的做法，打开一个事务，并将一个参数为 transaction 对象的回调函数作为参数传入 transaction 中，然后执行 executeSql()方法。方法中 SQL 查询语句和被点



击记录的 ID 作为两个参数传入。第 3 个参数是成功处理函数，但是不需要，所以指定为 null。而第 4 个参数，我们将其指定为一直使用的错误处理函数。这就是整个例子。虽然用了很多描述才完成这项功能，但实际上代码并不多。文件 123.js 只包含了差不多 100 行 JavaScript 代码。

下面演示了 123 与数据库交互的全部 JavaScript 代码清单：

```
var jQT=$.jQTouch(t
   icon: '123.png'
 });
 var db;
 $(document).ready(function(){
   $('#createEntry form') .submit(createEntry);
   $('#settings form').submit( saveSettings);
   $('#settings').bind('pageAnimationStart',loadSettings);
   $('#dates li a').click(function(){
   var dayOffset=this.id;
   var date=new Date();
   date.setDate(date.getDate()-dayOffset);
   sessionStorage.currentDate=date.getMonth()+l+'/'+
                           date.getDate() + '/l +
                           date.getFullYear(); .
                           refreshEntries();
   });
    var shortName = '123';
    var version = '1.0';
    var displayName = '123';
     var maxSize = 65536;
     db = openDatabase(shortName, version, displayName, maxSize);
        db.transaction(
      function(transaction) {
          transaction.executeSql(
                'CREATE TABLE IF NOT EXISTS entries. +
                '   (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, ' +
                '   date DATE NOT NULL, food TEXT NOT NULL, . +
                           .     calories INTEGER NOT NULL) ; '
                );
             }
        );
 });
 function loadSettings() t
          $( '#age').val(localStorage.age) ;
          $( '#budget').val(localStorage.budget) ;
          $( '#weight').val(localStorage.weight) ;
   }
 function saveSettings() {
          localStorage.age = $('#age') .val() ;
          localStorage.budget = $( '#budget') .val() ;
          localStorage.weight =  $( '#weight') .val() ;
          jQT.goBack() ;
     return false,
```




```
   }
 function createEntry(){
        var date=sessionStorage. currentDate .
         vat calories = $( '#calories') .val() i
         var food = $('#food').val();
         db.transaction(
         function(transaction) {
         transaction.executeSql(
                'INSERT INTO entries(date, calories,
                 food)VALUES(?, ?, ?);',
              [date, calories, foodl,
 function(){
                refreshEnt ries () ;
                jQT.goBack() ;
              },
 errorHandler
                  );
             }
       );
    return false,
   )
 function refreshEntries(){
       var currentDate=sessionStorage. currentDate ,
         $( '#date  hi') .text (currentDate) ;
          $( '#date  ul  li:gt (0) ' ) . remove () ;
db.transaction(
     fulclction(transaction) {
         transaction. executeSql(
            'SELECT* FROM entries WHERE date = ? ORDER BY food;',
             [currentDate],
              function (transaction, result){
              for(var i=o; i<result.rows.length; i++){
                     var row:result.rows.item(i);
                     var newEntryRow=$( '#entryTemplate') .clone()
                                  newEnt ryRow. removeAtt r( 'id ' )  ,
                                  newEnt ryRow. removeAttr( 'style')  ;
                                  newEntryRow.data( 'entryld',  row.id) ;
                                  newEntryRow.appendTo( '#date ul')i
                     newEntryRow.find('.label').text( row.food) ,
                     newEntryRow.find( ' .calo ries') .text ( row.calo ries) ;
                     newEnt ryRow.find ( '.delete').click(function() {
                 var clickedEntry = $(this).parent();
                 var clickedEntryId = clickedEntry.data('entryId);
                    deleteEnt ryById ( clickedEnt ryId)*
                        clickedEntry.slideUp();
            });
                     }
             },
 errorHandler
             );
          }
```

```
     );
 }
function deleteEntryById(id) {
        db.transaction (
    function(transaction) {
         transaction.executeSql('DE LETE FROM entries WHERE id=?;',
        Lidl, null, errorHandler);
         }
     );
 }
function errorHandler(transaction, error) {
   alert('Oops.Error was '+error.message+' (Code '+error.code+')');
     return.true,
}
```

第6章 WebKit浏览器详解

WebKit是Android系统的内置浏览器，是一个开源的浏览器网页排版引擎，包含WebCore排版引擎和JSCore引擎。WebCore和JSCore引擎来自于KDE项目的KHTML和KJS开源项目。Android平台的Web引擎框架采用了WebKit项目中的WebCore和JSCore部分，上层由Java语言封装，并且作为API提供给Android应用开发者，而底层使用WebKit核心库(WebCore和JSCore)进行网页排版。本章将详细讲解WebKit浏览器的基本知识，为读者步入本书后面知识的学习打下基础。

6.1 WebKit 的目录结构

Android 平台的 WebKit 模块分为 Java 层和 WebKit 库两个部分，其目录结构如表 6-1 所示。

表 6-1 WebKit的目录结构

目 录	说 明
Java 层(根目录是 device\java\android\android\webkit)	
BrowserFrame.java	BrowserFrame 对象是对 WebCore 库中的 Frame 对象的 Java 层封装，用于创建 WebCore 中定义的 Frame，以及为该 Frame 对象提供 Java 层回调方法
ByteArrayBuilder.java	ByteArrayBuilder 辅助对象，用于 byte 块链表的处理
CachLoader.java	URL Cache 载入器对象，该对象实现 StreadLoader 抽象基类，用于通过 CacheResult 对象载入内容数据
CacheManager.java	Cache 管理对象，负责 Java 层 Cache 对象管理
CacheSyncManager.java	Cache 同步管理对象，负责同步 RAM 和 Flash 之间的浏览器 Cache 数据。实际的物理数据操作在 WebSyncManager 对象中完成
CallbackProxy.java	该对象是用于处理 WebCore 与 UI 线程消息的代理类。当有 Web 事件产生时，WebCore 线程会调用该回调代理类，代理类会通过消息的方式通知 UI 线程，并且调用设置的客户对象的回调函数
CellList.java	CellList 定义图片集合中的 Cell，管理 Cell 图片的绘制、状态改变以及索引
CookieManager.java	根据 RFC2109 规范来管理 Cookies
CookieSyncManager.java	Cookies 同步管理对象，该对象负责同步 RAM 和 Flash 之间的 Cookies 数据。实际的物理数据操作在基类 WebSyncManager 中完成
DataLoader.java	数据载入器对象，用于载入网页数据
DateSorter.java	尚未使用
DownloadListener.java	下载侦听器接口
DownloadManagerCore.java	下载管理器对象，管理下载列表。该对象运行在 WebKit 的线程中，通过 CallbackProxy 对象与 UI 线程交互
FileLoader.java	文件载入器，将文件数据载入到 Frame 中
FrameLoader.java	Frame 载入器，用于载入网页 Frame 数据
HttpAuthHandler.java	Http 认证处理对象，该对象会作为参数传递给 BrowserCallback.displayHttpAuthDialog 方法，与用户交互
HttpDataTime.java	该对象是处理 HTTP 日期的辅助对象
JsConfirmResult.java	JS 确认请求对象



续表

目　录	说　明
JsPromptResult.java	JS 结果提示对象，用于向用户提示 JavaScript 运行结果
JsResult.java	JS 结果对象，用于实现用户交互
JWebCoreJavaBridge.java	用 Java 与 WebCore 库中 Timer 和 Cookies 对象交互的桥接代码
LoadListener.java	载入器侦听器，用于处理载入器侦听消息
Network.java	该对象封装网络连接逻辑，为调用者提供更为高级的网络连接接口
PanZoom.java	用于处理图片缩放、移动等操作
PanZoomCellList.java	用于保存移动、缩放图片的 Cell
SslErrorHandler.java	用于处理 SSL 错误消息
StreamLoader.java	StreamLoader 抽象类是所有内容载入器对象的基类。该类是通过消息方式控制的状态机，用于将数据载入到 Frame 中
TextDialog.java	用于处理 HTML 中文本区域叠加情况，可以使用标准的文本编辑而定义的特殊 EditText 控件
URLUtil.java	URL 处理功能函数，用于编码、解码 URL 字符串，以及提供附加的 URL 类型分析功能
WebBackForwardList.java	该对象包含 WebView 对象中显示的历史数据
WebBackForwardList-Client.java	浏览历史处理的客户接口类，所有需要接收浏览历史改变的类都需要实现该接口
WebChromeClient.java	Chrome 客户基类，Chrome 客户对象在浏览器文档标题、进度条、图标改变时会得到通知
WebHistoryItem.java	该对象用于保存一条网页历史数据
WebIconDataBase.java	图表数据库管理对象，所有的 WebView 均请求相同的图标数据库对象
WebSettings.java	WebView 的管理设置数据，该对象数据是通过 JNI 接口从底层获取
WebSyncManager.java	数据同步对象，用于 RAM 数据和 Flash 数据的同步操作
WebView.java	Web 视图对象，用于基本的网页数据载入、显示等 UI 操作
WebViewClient.java	Web 视图客户对象，在 Web 视图中有事件产生时，该对象可以获得通知
WebViewCore.java	该对象对 WebCore 库进行了封装，将 UI 线程中的数据请求发送给 WebCore 处理，并且通过 CallbackProxy 的方式，通过消息通知 UI 线程数据处理的结果
WebViewDatabase.java	该对象使用 SQLiteDatabase 为 WebCore 模块提供数据存取操作

6.2　WebKit 框架介绍

Android 平台的 WebKit 系统是由 Java 层和 WebKit 库两个部分组成，其中 Java 层负责与 Android 应用程序进行通信，而 WebKit 类库负责实际的网页排版处理。Java 层和 C 层库之间通过 JNI 和 Bridge 相互调用，如图 6-1 所示。

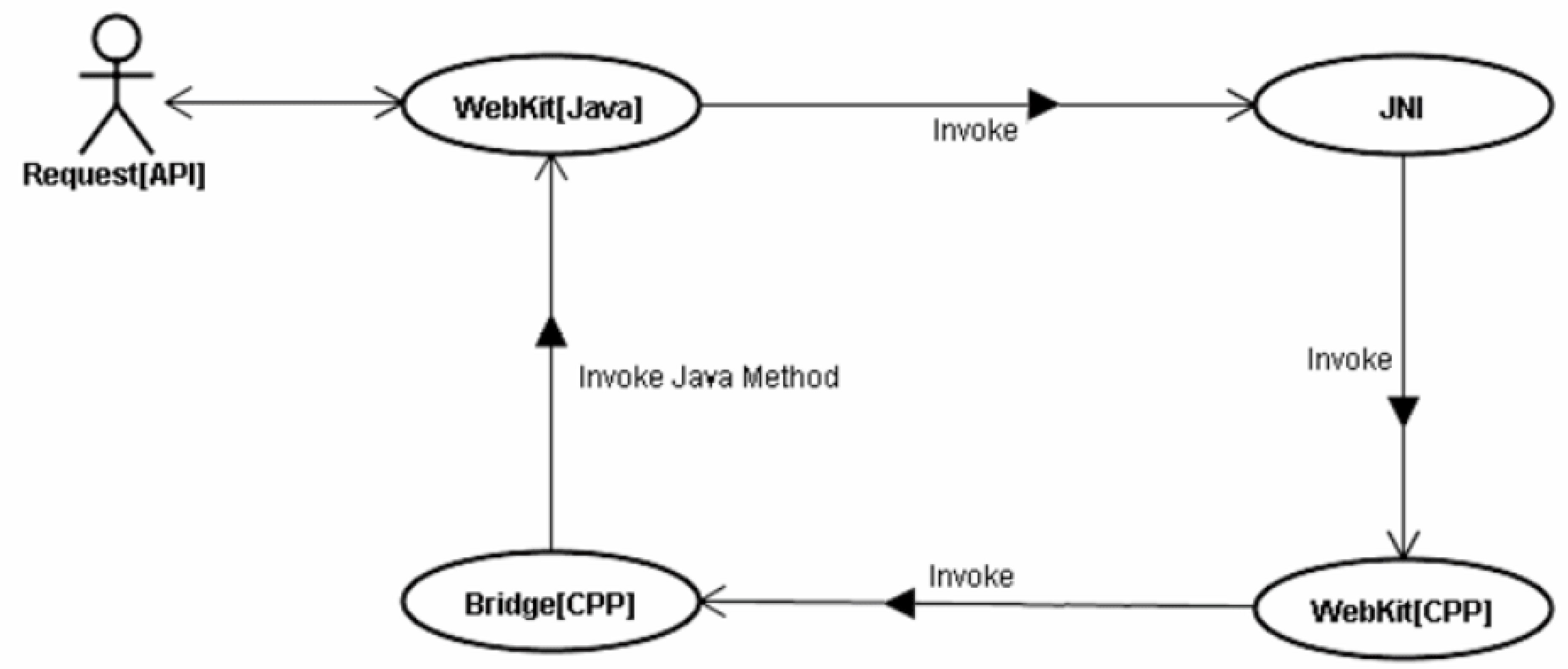


图 6-1　WebKit系统框架结构

6.2.1　Java层框架

1. 主要类

WebKit 模块的 Java 层一共由 41 个文件组成，其中主要类的具体说明如下。

1) WebView

类 WebView 是 WebKit 模块 Java 层的视图类，所有需要使用 Web 浏览功能的 Android 应用程序都要创建该视图对象显示和处理请求的网络资源。目前，WebKit 模块支持 HTTP、HTTPS、FTP 以及 JavaScript 请求。WebView 作为应用程序的 UI 接口，为用户提供了一系列的网页浏览、用户交互接口，客户程序通过这些接口访问 WebKit 核心代码。

WebView 是一个非常重要的类，能够实现和网络有关的很多功能，在本章后面的内容中将重点介绍。

2) WebViewDatabase

WebViewDatabase 是 WebKit 模块中针对 SQLiteDatabase 对象的封装，用于存储和获取运行时浏览器保存的缓冲数据、历史访问数据、浏览器配置数据等。该对象是一个单实例对象，通过 getInstance 方法获取 WebViewDatabase 的实例。WebViewDatabase 是 WebKit 模块中的内部对象，仅供 WebKit 框架内部使用。

3) WebViewCore

WebViewCore 类是 Java 层与 C 层 WebKit 核心库的交互类，客户程序调用 WebView 的网页浏览相关操作会转发给 BrowserFrame 对象。当 WebKit 核心库完成实际的数据分析和处理后会回调 WebViweCore 中定义的一系列 JNI 接口，这些接口会通过 CallbackProxy 将相关事件通知相应的 UI 对象。

4) CallbackProxy

CallbackProxy 是一个代理类，用于实现 UI 线程和 WebCore 线程之间的交互。类 CallbackProxy 定义了一系列与用户相关的通知方法，当 WebCore 完成相应的数据处理后会调用 CallbackProxy 类中对应的方法，这些方法通过消息方式间接调用相应处理对象的处理方法。

5) BrowserFrame

BrowserFrame 类负责 URL 资源的载入、访问历史的维护、数据缓存等操作，该类会通过 JNI 接口直接与 WebKit C 层库交互。

6) JWebCoreJavaBridge

类 JWebCoreJavaBridge 为 Java 层 WebKit 代码提供与 C 层 WebKit 核心部分的 Timer 和 Cookies 操作相关的方法。

7) DownloadManagerCore

类 DownloadManagerCore 是一个下载管理核心类，主要负责管理网络资源的下载，所有的 Web 下载操作均有该类统一管理。该类实例运行在 WebKit 线程当中，与 UI 线程的交互是通过调用 CallbackProxy 对象中相应的方法完成。

8) WebSettings

WebSettings 描述了 Web 浏览器访问相关的用户配置信息。

9) DownloadListener

DownloadListener 负责下载侦听接口，如果客户代码实现该接口，则在下载开始、失败、挂起、完成等情况下，DownloadManagerCore 对象会调用客户代码中实现的 DownloadListener 方法。

10) WebBackForwardList

WebBackForwardList 负责维护用户访问的历史记录，该类为客户程序提供操作访问浏览器历史数据的相关方法。

11) WebViewClient

在类 WebViewClient 中定义了一系列事件方法，如果 Android 应用程序设置了 WebViewClient 派生对象，则在页面载入、资源载入、页面访问错误等情况发生时，该派生对象的相应方法会被调用。

12) WebBackForwardListClient

WebBackForwardListClient 定义了对访问历史操作时可能产生的事件接口，当用户实现了该接口，则在操作访问历史(访问历史移除、访问历史清空等)时用户会得到通知。

13) WebChromeClient

类 WebChromeClient 定义了与浏览窗口修饰相关的事件。例如接收到 Title、Icon、进度变化时，WebChromeClient 的相应方法会被调用。

2. 数据载入器的设计理念

在 WebKit 系统的 Java 部分框架中，使用数据载入器来加载相应类型的数据，目前有 CacheLoader、DataLoader 以及 FileLoader 三类载入器，它们分别用于处理缓存数据、内存数据，以及文件数据的载入操作。Java 层(WebKit 模块)所有的载入器都从 StreamLoader 继承(其父类为 Handler)，由于 StreamLoader 类的基类为 Handler 类，因此在构造载入器时，会开启一个事件处理线程，该线程负责实际的数据载入操作，而请求线程通过消息的方式驱动数据的载入。图 6-2 描述了数据载入器相关类的类图结构。

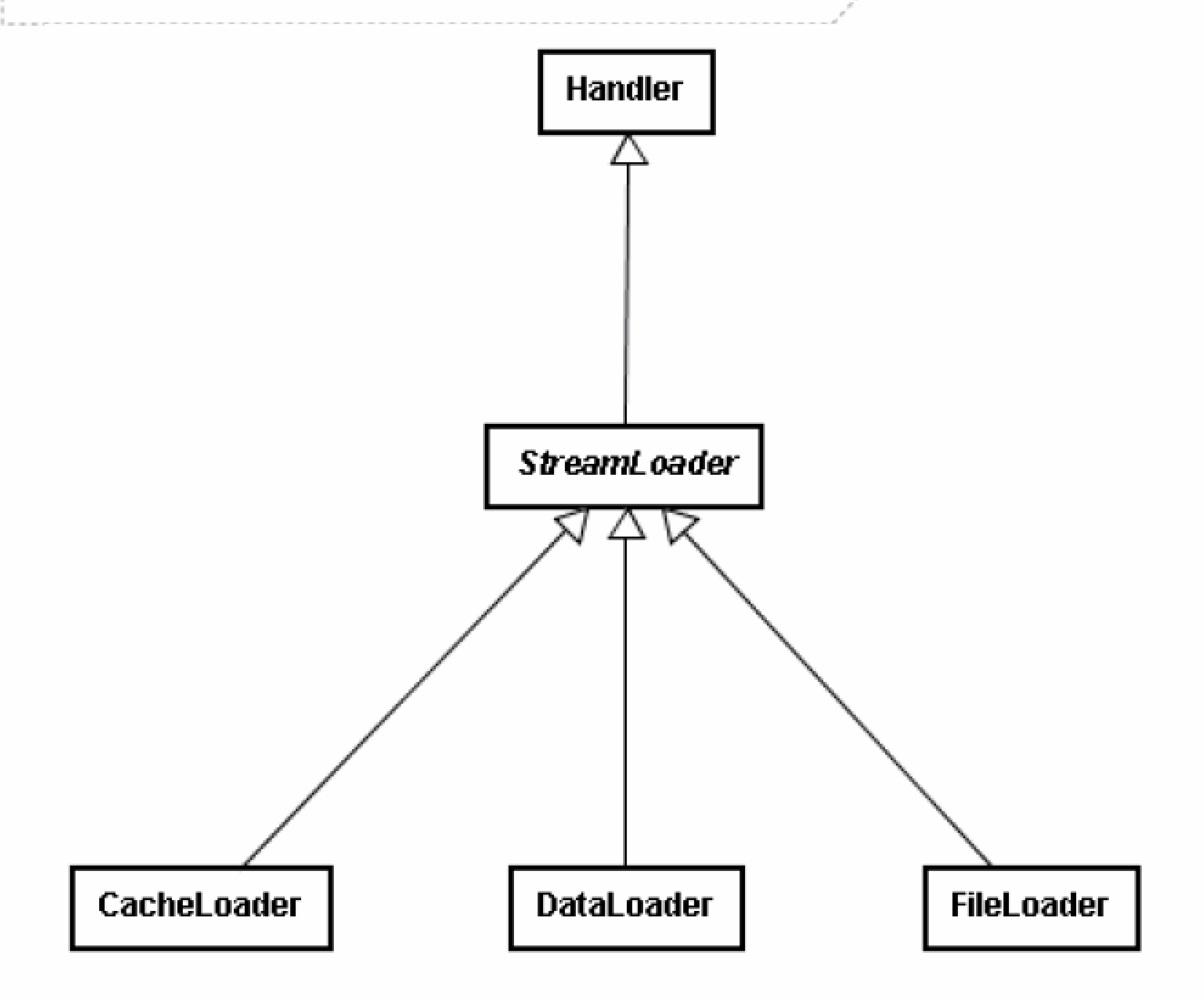


图 6-2　数据载入器的类图结构

在类 StreamLoader 中定义了以下 4 个不同的消息。

- MSG_STATUS：表示发送状态消息。
- MSG_HEADERS：表示发送消息头消息。
- MSG_DATA：表示发送数据消息。
- MSG_END：表示数据发送完毕消息。

在类 StreamLoader 中提供了两个抽象保护方法以及一个公有方法，其中保护方法 setupStreamAndSendStatus 用于构造与通信协议相关的数据流，以及向 LoadListener 发送状态。保护方法 buildHeaders 负责向子类提供构造特定协议消息头功能。所有载入器只有一个共有方法(load)，因此当需要载入数据时，只需调用该方法即可。与数据载入流程相关的类还有 LoaderListener 以及 BrowserFrame，当数据载入事件发生时，WebKit C 库会更新载入进度，并且会通知 BrowserFrame，BroserFrame 接收到进度条变更事件后会通过 CallbackProxy 对象，通知 View 类进度条数据变更。

6.2.2　C层框架

因为 C 层框架属于 Android 体系底层的知识，而我们本书主要讲解 Android 在 Java 层开发网络应用的知识，所以在此简要介绍 WebKit 系统 C 层框架的基本知识，只简单分析 C 层框架中各个类之间的关系。读者了解了这些类之间的关系和原理后，当在 Java 层中开发应用时即可“游刃有余”。

1. C层类和Java层的关系

1) BrowserFrame

与 BrowserFrame Java 类相对应的 C++类为 FrameBridge，该类为 Dalvik 虚拟机回调 BrowserFrame 类中定义的本地方法进行了封装。与 BrowserFrame 中回调函数(Java 层)相对应的 C 层结构定义代码如下。



```
struct FrameBridge::JavaBrowserFrame
{
    JavaVM*   mJVM;
    jobject   mObj;
    jmethodID mStartLoadingResource;
    jmethodID mLoadStarted;
    jmethodID mUpdateHistoryForCommit;
    jmethodID mUpdateCurrentHistoryData;
    jmethodID mReportError;
    jmethodID setTitle;
    jmethodID mWindowObjectCleared;
    jmethodID mDidReceiveIcon;
    jmethodID mUpdateVisiteHistory;
    jmethodID mHandleUrl;
    jmethodID mCreateWindow;
    jmethodID mCloseWindow;
    jmethodID mDecidePolicyForFormResubmission;
};
```

上述结构作为 FrameBridge(C 层)的一个成员变量(mJavaFrame)，在 FrameBridge 构造函数中，用 BrowserFrame(Java 层)类的回调方法的偏移量初始化 JavaBrowserFrame 结构的各个域。初始化后，当 WebCore(C 层)在剖析网页数据时，有 Frame 相关的资源改变，比如 Web 页面的主题变化，则会通过 mJavaFrame 结构，调用指定 BrowserFrame 对象的相应方法，通知 Java 层处理。

2) JWebCoreJavaBridge

与 JWebCoreJavaBridge 对象相对应的 C 层对象为 JavaBridge，JavaBridge 对象继承了 TimerClient 和 CookieClient 类，负责 WebCore 中的定时器和 Cookie 管理。与 Java 层 JWebCoreJavaBridge 类中方法偏移量相关的是 JavaBridege 中的几个成员变量，在构造 JavaBridge 对象时，会初始化这些成员变量，之后有 Timer 或者 Cookies 事件产生，WebCore 会通过这些 ID 值，回调对应 JWebCoreJavaBridge 的相应方法。

3) LoadListener

与 LoadListener 对象相关的 C 层结构是 struct resourceloader_t，该结构保存了 LoadListener 对象 ID、CancelMethod ID 以及 DownloadFileMethod ID 值。当有 Cancel 或者 Download 事件产生，WebCore 会回调 LoadListener 类中的 CancelMethod 或者 DownloadFileMethod。

4) WebViewCore

与 WebViewCore 相关的 C 类是 WebCoreViewImpl，WebCoreImplView 类有个 JavaGlue 对象作为成员变量，在构建 WebCoreViewImpl 对象时，用 WebViewCore(Java 层)中的方法 ID 值初始化该成员变量，并且会将构建的 WebCoreViewImpl 对象指针赋值给 WebViewCore(Java 层)的 mNativeClass，这样将 WebViewCore(Java 层)和 WebViewCoreImpl(C 层)关联起来。

5) WebSettings

与 WebSettings 相关的 C 层结构是 struct FieldIds，该结构保存了 WebSettings 类中定义的属性 ID 以及方法 ID，在 WebCore 初始化(WebViewCore 的静态方法中使用

System.loadLibrary 载入)时会设置这些方法和属性的 ID 值。

6) WebView

与 WebView 相关的 C 层类是 WebViewNative，该类中的 mJavaGlue 中保存着 WebView 中定义的属性和方法 ID，在 WebViewNative 构造方法中初始化，并且将构造的 WebViewNative 对象的指针赋值给 WebView 类的 mNativeClass 变量，这样 WebView 和 WebViewNative 对象建立了关系。

2. 与Java层相关的C层类

下面总结与 Java 层相关的 C 层类，具体信息如下。

- ChromeClientAndroid：该类主要处理 WebCore 中与 Frame 装饰相关的操作。例如设置状态栏、滚动条、JavaScript 脚本提示框等。当浏览器中有相关事件产生时，ChromeClientAndroid 类的相应方法会被调用，该类会将相关的 UI 事件通过 Bridge 传递给 Java 层，由 Java 层负责绘制以及用户交互方面的处理。
- EditorClientAndroid：该类负责处理页面中文本相关的处理，比如文本输入、取消、输入法数据处理、文本粘贴、文本编辑等操作。不过目前该类只对按键相关的时间进行了处理，其他操作均未支持。
- ContextMenuClient：该类提供页面相关的功能菜单，比如图片拷贝、朗读、查找等功能。但是，目前项目中未实现具体功能。
- DragClient：该类定义了与页面拖曳相关的处理，但是目前该类没有实现具体功能。
- FrameLoaderClientAndroid：该类提供与 Frame 加载相关的操作，当用户请求加载一个页面时，WebCore 分析完网页数据后，会通过该类调用 Java 层的回调方法，通知 UI 相关的组件处理。
- InspectorClientAndroid：该类提供与窗口相关的操作，比如窗口显示、关闭窗口、附加窗口等。不过目前该类的各个方法均为空实现。
- Page：该类提供与页面相关的操作，比如网页页面的前进、后退等操作。
- FrameAndroid：该类为 Android 提供 Frame 管理。
- FrameBridge：该类对 Frame 相关的 Java 层方法进行了封装，当有 Frame 事件产生时，WebCore 通过 FrameBridge 回调 Java 的回调函数，完成用户交互过程。
- AssetManager：该类为浏览器提供本地资源访问功能。
- RenderSkinAndroid：该类与控件绘制相关，所有的需绘制控件都必须从该类派生，目前 WebKit 模块中有 Button、Combo、Radio 三类控件。

上述类会在 Java 层请求创建 Web Frame 的时候被建立。

6.3 WebKit 操作

经过本章前面内容的学习，相信大家已经基本了解了 WebKit 系统的运作原理和各层中各个主要类的功能。本节将简单介绍和 WebKit 相关的基本操作知识，为读者步入本书后面知识的学习打下基础。



6.3.1　WebKit初始化

在 Android SDK 中提供了 WebView 类，使用该类可以提供客户化浏览显示功能。如果客户需要加入浏览器的支持，可将该类的实例或者派生类的实例作为视图，调用 Activity 类的 setContentView 显示给用户。当客户代码中第一次生成 WebView 对象时，会初始化 WebKit 库(包括 Java 层和 C 层两个部分)，之后用户可以操作 WebView 对象完成网络或者本地资源的访问。

WebView 对象的生成主要涉及 3 个类：CallbackProxy、WebViewCore 以及 WebViewDatabase。其中 CallbackProxy 对象为 WebKit 模块中 UI 线程和 WebKit 类库提供交互功能，WebViewCore 是 WebKit 的核心层，负责与 C 层交互以及 WebKit 模块 C 层类库初始化，而 WebViewDatabase 为 WebKit 模块运行时缓存、数据存储提供支持。

初始化的过程就是使用 WebView 创建 CallbackProxy 对象和 WebViewCore 对象的过程。WebKit 模块初始化流程如下。

(1) 调用 System.loadLibrary 载入 WebCore 相关类库(C 层)。

(2) 如果是第一次初始化 WebViewCore 对象，创建 WebCoreThread 线程。

(3) 创建 EventHub 对象，处理 WebViewCore 事件。

(4) 获取 WebIconDatabase 对象实例。

(5) 向 WebCoreThread 发送初始化消息。

根据上述流程，假如我们要获取 WebViewDatabase 实例，则可以按照下面的步骤来实现。

(1) 调用 System.loadLibrary 方法载入 WebCore 相关类库，该过程由 Dalvik 虚拟机完成，它会从动态链接库目录中寻找 libWebCore.so 类库，载入到内存中，并且调用 WebKit 初始化模块的 JNI_OnLoad 方法。WebKit 模块的 JNI_OnLoad 方法完成以下初始化操作。

- 初始化 FrameBridge[register_android_webcore_framebridge]：初始化 gFrameAndroidField 静态变量，以及注册 BrowserFrame 类中的本地方法表。
- 初始化 JavaBridge[register_android_webcore_javabridge]：初始化 gJavaBridge.mObject 对象，以及注册 JWebCoreJavaBridge 类中的本地方法。
- 初始化资源 Loader[register_android_webcore_resource_loader]：初始化 gResourceLoader 静态变量，以及注册 LoadListener 类的本地方法。
- 初始化 WebViewCore[register_android_webkit_webviewcore]：初始化 gWebCoreViewImplField 静态变量，以及注册 WebViewCore 类的本地方法。
- 初始化 WebHistory[register_android_webkit_webhistory]：初始化 gWebHistoryItem 结构，以及注册 WebBackForwardList 和 WebHistoryItem 类的本地方法。
- 初始化 WebiconDatabase[register_android_webkit_webicondatabase]：注册 WebIconDatabase 类的本地方法。
- 初始化 WebSettings[register_android_webkit_websettings]：初始化 gFieldIds 静态变量，以及注册 WebSettings 类的本地方法。
- 初始化 WebView[register_android_webkit_webview]：初始化 gWebViewNativeField

静态变量，以及注册 WebView 类的本地方法。

(2) 实现 WebCoreThread 初始化，该初始化只在第一次创建 WebViewCore 对象时完成，当用户代码第一次生成 WebView 对象，会在初始化 WebViewCore 类时创建 WebCoreThread 线程，该线程负责处理 WebCore 初始化事件。此时 WebViewCore 构造函数会被阻塞，直到一个 WebView 初始化请求完毕时，会在 WebCoreThread 线程中唤醒。

(3) 创建 EventHub 对象，该对象处理 WebView 类的事件，当 WebCore 初始化完成后会向 WebView 对象发送事件，WebView 类的 EventStub 对象处理该事件，并且完成后续初始化工作。

(4) 获取 WebIconDatabase 对象实例。

(5) 向 WebViewCore 发送 INITIALIZE 事件，并且将 this 指针作为消息内容传递。WebView 类主要负责处理 UI 相关的事件，而 WebViewCore 主要负责与 WebCore 库交互。在运行时期，UI 线程和 WebCore 数据处理线程是运行在两个独立的线程当中。WebCoreThread 线程接收到 INITIALIZE 事件后，会调用消息对象参数的 initialize 方法，而后唤醒阻塞的 WebViewCore Java 线程(该线程在 WebViewCore 的构造函数中被阻塞)。不同的 WebView 对象实例有不同的 WebViewCore 对象实例，因此通过消息的方式可以使得 UI 线程和 WebViewCore 线程解耦合。WebCoreThread 的事件处理函数，处理 INITIALIZE 消息时，调用的是不同 WebView 中 WebViewCore 实例的 initialize 方法。WebViewCore 类中的 initialize 方法中会创建 BrowserFrame 对象(该对象管理整个 Web 窗体，以 Frame 相关事件)，并且向 WebView 对象发送 WEBCORE_INITIALIZED_MSG_ID 消息。WebView 消息处理函数能够根据其参数来初始化指定 WebViewCore 对象，并且能够更新 WebViewCore 的 Frame 缓冲。

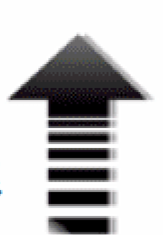
6.3.2 载入数据

1. 载入网络数据

在代码中可以使用 WebView 类的 loadUrl 方法，请求访问指定的 URL 网页数据。WebView 对象中保存着 WebViewCore 的引用，由于 WebView 属于 UI 线程，而 WebViewCore 属于后台线程，因此 WebView 对象的 loadUrl 被调用时，会通过消息的方式将 URL 信息传递给 WebViewCore 对象，该对象会调用成员变量 mBrowserFrame 的 loadUrl 方法，进而调用 WebKit 库完成数据的载入。

在载入网络数据时，此功能分别由 Java 层和 C 层共同完成，其中 Java 层负责完成用户交互、资源下载等操作，而 C 层主要完成数据分析(建立 DOM 树、分析页面元素等)操作。由于 UI 线程和 WebCore 线程运行在不同的两个线程中，因此当用户请求访问网络资源时，通过消息的方式向 WebViewCore 对象发送载入资源请求。

在 Java 层的 WebKit 模块中，所有与资源载入相关的操作都是由 BrowserFrame 类中对应的方法完成，这些方法是本地方法，会直接调用 WebCore 库的 C 层函数完成数据载入请求，以及资源分析等操作。C 层的 FrameLoader 类是浏览框架的资源载入器，该类负责检查访问策略以及向 Java 层发送下载资源请求等功能。在 FrameLoader 中，当用户请求网络资源时，经过一系列的策略检查后会调用 FrameBridge 的 startLoadingResource 方法，该



方法会回调 BrowserFrame(Java)类的 startLoadingResource 方法，完成网络数据的下载，然后类 BrowserFrame(Java)的方法 startLoadingResource 会返回一个 LoadListener 的对象，FrameLoader 会删除原有的 FrameLoader 对象，将 LoadListener 对象封装成 ResourceLoadHandler 对象，并且将其设置为新的 FrameLoader。到此完成了一次资源访问请求，接下来库 WebCore 会根据资源数据进行分析和构建 DOM，以及构建相关的数据结构。

2. 载入本地数据

所谓本地数据是指以“data://”开头的 URL，载入本地数据的过程和载入网络数据的方法一样，只不过在执行 FrameLoader 类的 executeLoad 方法时，会根据 URL 的 SCHEME 类型区分，调用 DataLoader 的 requestUrl 方法，而不是调用 handleHTTPLoad 建立实际的网络通信连接。

3. 载入文件数据

所谓文件数据是指以“file://”开头的 URL，载入文件数据的基本流程与网络数据载入流程基本一致，不同的是在运行 FrameLoader 类的 executeLoad 方法时，根据 SCHEME 类型，调用 FileLoader 的 requestUrl 方法来完成数据加载。

6.3.3　刷新绘制

当用户拖动滚动条、有窗口遮盖或者有页面事件触发时都会向 WebViewCore(Java 层)对象发送背景重绘消息，该消息会引起网页数据的绘制操作。WebKit 的数据绘制可能出于效率上的考虑，没有通过 Java 层，而是直接在 C 层使用 SGL 库完成。与 Java 层图形绘制相关的 Java 对象有 3 个，具体说明如下。

- Picture 类：该类对 SGL 封装，其中变量 mNativePicture 实际上是保存着 SkPicture 对象的指针。WebViewCore 中定义了两个 Picture 对象，当作双缓冲处理，在调用 webKitDraw 方法时，会交换两个缓冲区，加速刷新速度。
- WebView 类：该类接受用户交互相关的操作，当有滚屏、窗口遮盖、用户点击页面按钮等相关操作时，WebView 对象会与之相关的 WebViewCore 对象发送 VIEW_SIZE_CHANGED 消息。当 WebViewCore 对象接收到该消息后，将构建时建立的 mContentPictureB 刷新到屏幕上，然后将 mContentPictureA 与之交换。
- WebViewCore 类：该类封装了 WebKit C 层代码，为视图类提供对 WebKit 的操作接口，所有对 WebKit 库的用户请求均由该类处理，并且该类还为视图类提供了两个 Picture 对象，用于图形数据刷新。

例如我们在拖曳 Web 页面，当用户使用手指点击触摸屏并且移动手指时会引发 touch 事件，Android 平台会将 touch 事件传递给最前端的视图响应(dispatchTouchEvent 方法处理)。在 WebView 类中定义了 5 种 touch 模式，在手指拖动 Web 页面的情况下，会触发 mMotionDragMode，并且会调用 View 类的 scrollBy 方法，触发滚屏事件以及使视图无效(重绘，会调用 View 的 onDraw 方法)。WebView 视图中的滚屏事件由 onScrollChanged 方法响应，该方法向 WebViewCore 对象发送 SET_VISIBLE_RECT 事件。

WebViewCore 对象接收到 SET_VISIBLE_RECT 事件后，将消息参数中保存的新视图的矩形区域大小传递给 nativeSetVisibleRect 方法，通知 WebCoreViewImpl 对象(C 层)视图矩形变更(WebCoreViewImpl::setVisibleRect 方法)。在 setVisibleRect 方法中，会通过虚拟机调用 WebViewCore 的 contentInvalidate 方法，该方法会引发 webkitDraw 方法的调用(通过 WEBKIT_DRAW 消息)。在方法 webkitDraw 中，首先会将 mContentPictureB 对象传递给本地方法 nativeDraw 绘制，然后将 mContentPictureB 的内容与 mContentPictureA 的内容互换。在这里 mContentPictureA 缓冲区是供给 WebViewCore 的 draw 方法使用，如果用户选择某个控件，绘制焦点框时 WebViewCore 对象的 draw 方法会被调用，绘制的内容保存在 mContentPictureA 中，之后会通过 Canvas 对象(Java 层)的 drawPicture 方法将其绘制到屏幕上，而 mContentPictureB 缓冲区是用于 built 操作的，nativeDraw 方法中首先会将传递的 mContentPictureB 对象数据重置，而后在重新构建的 mContentPictureB 画布上，将层上相关的元素绘制到该画布上，然后将 mContentPictureB 和 mContentPictureA 的内容互换，这样一次重绘事件产生时(会调用 WebView.onDraw 方法)会将 mContentPictureA 的数据使用 Canvas 类的 drawPicture 绘制到屏幕上。当 webkitDraw 方法将 mContentPictureA 与 mContentPictureB 指针对调后，会向 WebView 对象发送 NEW_PICTURE_MSG_ID 消息，该消息会引发 WebViewCore 的 VIEW_SIZE_CHANGED 消息的产生，并且会使当前视图无效，产生重绘事件(invalidate())，引发 onDraw 方法的调用，完成一次网页数据的绘制过程。

6.4 WebView 类详解

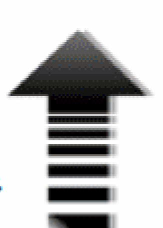
在本章前面的内容中曾经提到过，WebView 是一个非常重要的类，能够实现和网络有关的很多功能。WebView 能加载显示网页，可以将其视为一个浏览器，使用 WebKit 渲染引擎来加载显示网页。在本节的内容中，将详细讲解 WebView 的基本知识。

6.4.1 WebView概述

通过 WebView 可以滚动 Web 浏览器或简单地显示您在网上活动的某些内容。WebView 采用 WebKit 渲染引擎来显示网页的方法，包括向前和向后导航的历史、放大和缩小、执行文本搜索和是否启用内置的变焦。

WebView 中的主要方法如下。

- addJavascriptInterface(Object obj, StringinterfaceName)：使用此函数来绑定一个对象的 JavaScript，该方法可以访问 JavaScript。
- loadData(String data, String mimeType, Stringencoding)：此方法经常出现乱码，尽量少用。
- loadDataWithBaseURL(String baseUrl, String data, String mimeType,String encoding, StringhistoryUrl)：加载到 WebView 给定的数据，以此为基础内容的网址提供的网址。
- capturePicture()：捕捉当前 WebView 的图片。



- clearCache(boolean includeDiskFiles)：清除资源的缓存。
- destroy()：销毁此 WebView。
- setDefaultFontSize()：设置字体。
- setDefaultZoom()：设置屏幕的缩放级别。

在 Android 的所有控件中，WebView 的功能最强大，它作为直接从 android.webkit.Webview 实现的类，可以拥有浏览器所有的功能，WebView 可以让开发人员从 Java 转向“HTML+JS”这样的方式。如果具备了 Ajax 技术，就可以方便通过这种方式配合远端 Server 来实现一些内容。

从 Android 2.2 中开始加入了 Adobe Flash Player 功能，我们可以通过如下代码设置允许 Gears 插件来实现网页中的 Flash 动画显示。

```
WebView.getSettings().setPluginsEnabled(true);
```

通过使用 WebView，可以帮助我们设计内嵌专业的浏览器，相对于部分以省流量需要服务器中转的那种 HTML 解析器来说有本质的区别，因为它们没有 JavaScript 脚本解析器，所以不会有什么太大的发展空间。

1. 访问网页

通过 loadUrl()方法可以访问网页，例如下面的代码：

```
wb=(WebView)findViewById(R.id.wb);
wb.loadUrl(url);
```

2. 设置属性

对于浏览器的设置，可以通过 WebSettings 来设置 WebView 的一些属性和状态等，例如下面的代码：

```
WebSettingswebSettings=mWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
//设置可以访问文件
webSettings.setAllowFileAccess(true);
//设置支持缩放
webSettings.setBuiltInZoomControls(true);
```

3. WebViewClient和WebChromeClient

WebViewClient 和 WebChromeClient 可以看作是辅助 WebView 管理网页中各种通知、请求等事件以及 JavaScript 时间的两个类。

1) WebViewClient

利用 WebView 的 setWebViewClient()方法可以指定一个 WebViewClient 对象，通过覆盖该类的方法来辅助 WebView 浏览网页。例如下面的代码：

```
mWebView.setWebViewClient(newWebViewClient()
  {
  publicbooleanshouldOverrideUrlLoading(WebViewview,Stringurl)
  {
  view.loadUrl(url);
```

```
returntrue;
}
@Override
publicvoidonPageFinished(WebViewview,Stringurl)
{
super.onPageFinished(view,url);
}
@Override
publicvoidonPageStarted(WebViewview,Stringurl,Bitmapfavicon)
{
super.onPageStarted(view,url,favicon);
}
});
```

2) WebChromeClient

对于网页中使用的 JavaScript 脚本语言，便可以使用 WebChromeClient 类处理 JS 事件，如对话框加载进度等，例如下面的代码：

```
mWebView.setWebChromeClient(newWebChromeClient(){
@Override
//处理 JavaScript 中的 alert
publicbooleanonJsAlert(WebViewview,Stringurl,Stringmessage,
finalJsResultresult)
{
//构建一个 Builder 来显示网页中的对话框
Builderbuilder=newBuilder(Activitythis);
builder.setTitle("提示对话框");
builder.setMessage(message);
builder.setPositiveButton(android.R.string.ok,
newAlertDialog.OnClickListener(){
publicvoidonClick(DialogInterfacedialog,intwhich){
//单击“确定”按钮之后，继续执行网页中的操作
result.confirm();
}
});
builder.setCancelable(false);
builder.create();
builder.show();
returntrue;
};
```

6.4.2　实现WebView的两种方式

WebView 能够以加载的方式显示网页，可以将其视为一个浏览器。WebView 使用 WebKit 渲染引擎加载显示网页。在开发应用中有以下两种实现 WebView 的方法。

1. 第一种实现方法

(1) 在 Activity 中实例化 WebView 组件。



```
WebView webView = new WebView(this);
```

(2) 调用 WebView 的 loadUrl()方法，设置 WebView 要显示的网页。

- 如果显示互联网则使用：

```
webView.loadUrl("http://www.google.com");
```

- 如果显示本地文件则使用：

```
webView.loadUrl("file:///android asset/XX.html");//本地文件存放在“assets”
文件中
```

(3) 调用 Activity 的 setContentView()方法来显示网页视图。

(4) 用 WebView 点链接看了很多页以后为了让 WebView 支持回退功能，需要覆盖 Activity 类的 onKeyDown()方法，如果不做任何处理，点击系统回退键，整个浏览器会调用 finish()而结束自身，而不是回退到上一页面。

(5) 在 AndroidManifest.xml 文件中添加权限，否则会出现“Web page not available”错误。

```
  <uses-permission android:name="android.permission.INTERNET" />
```

接下来我们看一个使用上述方法的演示代码。

首先编写程序文件 MainActivity.java，具体代码如下：

```
package com.android.webview.activity;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.webkit.WebView;

public class MainActivity extends Activity {
    private WebView webview;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //实例化 WebView 对象
        webview = new WebView(this);
        //设置 WebView 属性，能够执行 JavaScript 脚本
        webview.getSettings().setJavaScriptEnabled(true);
        //加载需要显示的网页
        webview.loadUrl("http://www.51cto.com/");
        //设置 Web 视图
        setContentView(webview);
    }

    @Override
    //设置回退
    //覆盖 Activity 类的 onKeyDown(int keyCoder,KeyEvent event)方法
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if ((keyCode == KeyEvent.KEYCODE_BACK) && webview.canGoBack()) {
```

```
            webview.goBack(); //goBack()表示返回 WebView 的上一页面
            return true;
        }
        return false;
}
```

然后在文件 AndroidManifest.xml 中添加如下 INTERNET 权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

2. 第二种实现方式

(1) 在布局文件中声明 WebView。

(2) 在 Activity 中实例化 WebView。

(3) 调用 WebView 的 loadUrl( )方法，设置 WebView 要显示的网页。

(4) 为了让 WebView 能够响应超链接功能，调用 setWebViewClient()方法，设置 WebView 视图

(5) 用 WebView 点链接看了很多页以后为了让 WebView 支持回退功能，需要覆盖 Activity 类的 onKeyDown()方法，如果不做任何处理，点击系统回退键，整个浏览器会调用 finish()而结束自身，而不是回退到上一页面。

(6) 在文件 AndroidManifest.xml 中添加如下权限，否则出现“Web page not available”错误。

```
<uses-permission android:name="android.permission.INTERNET"/>
```

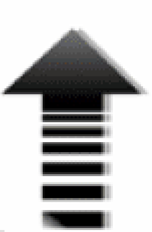
接下来我们看一个使用上述方法的演示代码。

首先编写程序文件 MainActivity.java，具体代码如下：

```
package com.android.webview.activity;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class MainActivity extends Activity {
    private WebView webview;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        webview = (WebView) findViewById(R.id.webview);
        //设置 WebView 属性，能够执行 JavaScript 脚本
        webview.getSettings().setJavaScriptEnabled(true);
        //加载需要显示的网页
        webview.loadUrl("http://www.51cto.com/");
        //设置 Web 视图
        webview.setWebViewClient(new HelloWebViewClient ());
    }
```



```
    @Override
    //设置回退
    //覆盖 Activity 类的 onKeyDown(int keyCoder,KeyEvent event)方法
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if ((keyCode == KeyEvent.KEYCODE_BACK) && webview.canGoBack()) {
            webview.goBack(); //goBack()表示返回 WebView 的上一页面
            return true;
        }
        return false;
    }

    //Web 视图
    private class HelloWebViewClient extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            view.loadUrl(url);
            return true;
        }
    }
}
```

然后编写布局文件 main.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <WebView
        android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        />
</LinearLayout>
```

最后在文件 AndroidManifest.xml 中添加 INTERNET 权限，代码如下：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

6.4.3　WebView的常见功能

WebView 有以下几个常见功能。

(1) 背景设置。例如下面的代码：

```
WebView.setBackgroundColor(0);//先设置背景色为 transparent
WebView.setBackgroundResource(R.drawable.yourImage);//然后设置背景图片
```

(2) 获得 WebView 网页加载初始化和完成事件。基本步骤如下。

① 创建一个自己的、继承于 WebViewClient 类的 WebViewClient，例如

WebViewClient。

② 重载 onPageFinished()方法(WebView 加载完成会调用这个方法)。

③ 通过方法 webView.setWebViewClient()关联 WebViewClient 与 WebView。

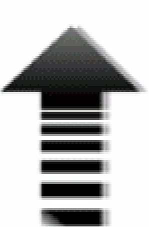
例如下面的代码：

```
mWebView.setWebViewClient(new WebViewClient()
{
 @Override
public void onPageFinished(WebView view, String url)
{
//结束
super.onPageFinished(view, url);
}
  @Override
  public void onPageStarted(WebView view, String url, Bitmap favicon)
  {
  //开始
  super.onPageStarted(view, url, favicon);
  }
});
```

如果需要监视加载进度，则需要创建一个 WebChromeClient 类，并重载方法 onProgressChanged，再进行 webview.setWebChromeClient(new MyWebChromeClient())即可。例如下面的代码：

```
class MyWebChromeClient extends WebChromeClient {
@Override
public void onProgressChanged(WebView view, int newProgress) {
// TODO Auto-generated method stub
super.onProgressChanged(view, newProgress);
}
}
public class WebPageLoader extends Activity {
    final Activity activity = this;


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.getWindow().requestFeature(Window.FEATURE_PROGRESS);
        setContentView(R.layout.main);
        WebView webView = (WebView) findViewById(R.id.webView);
        webView.getSettings().setJavaScriptEnabled(true);
        webView.getSettings().setSupportZoom(true);
        webView.setWebChromeClient(new WebChromeClient() {
            public void onProgressChanged(WebView view, int progress) {
                activity.setTitle("Loading...");
                activity.setProgress(progress * 100);
                if (progress == 100)
                    activity.setTitle(R.string.app_name);
```



```
            }
        });
        webView.setWebViewClient(new WebViewClient() {
            public void onReceivedError(WebView view, int errorCode,
                String description, String failingUrl) { // Handle the error
            }


            public boolean shouldOverrideUrlLoading(WebView view, String url) {
                view.loadUrl(url);
                return true;
            }
        });
        webView.loadUrl("http://www.sohu.com");
    }
}
```

(3) 使用 WebView 阅读 PDF 文件。

Android 本身不支持打开 PDF 文件，其实 Google 提供了在线解析 PDF 的方法，即使用 WebView 来实现。例如下面的代码：

```
WebView webview = (WebView) findViewById(R.id.wv);
webview.getSettings().setJavaScriptEnabled(true);
String pdf ="http://www.*****.pdf";
webview.loadUrl("http://docs.google.com/gview?embedded=true&url=" + pdf);
```

(4) 当用 WebView 加载网页时在标题栏上显示加载进度。

这个功能很容易理解，如图 6-3 所示。当在使用 WebView 加载网页时，可以在标题栏显示加载进度，这样做的目的是更加友好地提示用户。



图 6-3　加载网页

例如下面的代码：

```
public class ProgressTest extends Activity{
final Activity context = this;

@Override
public void onCreate(Bundle b) {
   super.onCreate(b);
   requestWindowFeature(Window.FEATURE_PROGRESS);//让进度条显示在标题栏上
   setContentView(R.layout.main);
   WebView webview = (WebView)findViewById(R.id.webview);
   webview.setWebChromeClient(new WebChromeClient() {
           public void onProgressChanged(WebView view, int progress) {
```




```
            //Activity 和 WebView 根据加载程度决定进度条的进度大小
           //当加载到 100%的时候进度条自动消失
            context.setProgress(progress * 100);
        }
    });
   webview.loadUrl(url);
}
```

其实上述功能在 Android 开发中十分常见，当前主流的开发模式是“WebView+ProgressDialog”。再看下面演示代码的实现过程。

首先编写一个名为 webview.xml 的布局文件，代码如下：

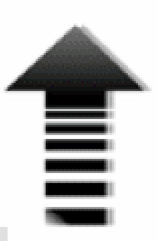
```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout width="fill parent"
  android:layout height="fill parent">
    <WebView android:id="@+id/webview"
        android:layout width="fill parent"
        android:layout height="fill parent"/>
</LinearLayout>
```

然后编写一个名为 WebViewActivity.java 的工程文件，主要代码如下：

```
public class WebViewActivity extends Activity{
    private WebView webView;

    private AlertDialog alertDialog;
    private ProgressDialog progressBar;
    //jQuery datatables 使用
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.webview);
        //加载 WebView
        initWebView();
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if(keyCode == KeyEvent.KEYCODE BACK && webView.canGoBack()){
            webView.goBack();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
    class MyWebViewClient extends WebViewClient{
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url)
{
            view.loadUrl(url);
            return true;
```



```
        }

        @Override
        public void onPageFinished(WebView view, String url) {
            if(progressBar.isShowing()){
                progressBar.dismiss();
            }
        }

        @Override
        public void onReceivedError(WebView view, int errorCode,
                String description, String failingUrl) {
                    Toast.makeText(WebViewActivity.this, "网页加载出错! ",
                    Toast.LENGTH LONG);

            alertDialog.setTitle("ERROR");
            alertDialog.setMessage(description);
            alertDialog.setButton("OK", new
                  DialogInterface.OnClickListener(){
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    // TODO Auto-generated method stub
                }
            });
            alertDialog.show();
        }
    }

    protected void initWebView(){
        //设计进度条
        progressBar = ProgressDialog.show(WebViewActivity.this, null,
            "正在进入网页，请稍后…");
        //获得 WebView 组件
        webView = (WebView) this.findViewById(R.id.webview);

        webView.getSettings().setJavaScriptEnabled(true);

        webView.loadUrl("http://www.baidu.com");

        alertDialog = new AlertDialog.Builder(this).create();

        //设置视图客户端
        webView.setWebViewClient(new MyWebViewClient());
    }
}
```

最后，在文件 AndroidManifest.xml 中添加访问互联网的权限，否则不能显示。代码如下：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

上述过程就是基于“WebView+ProgressDialog”开发模式的过程。

(5) 使用 WebView 调用拨号键盘。例如下面的代码：

```
wv.setWebViewClient(new WebViewClient(){
          public boolean shouldOverrideUrlLoading(WebView view,String url){
              //当有新连接时，使用当前的 WebView
              view.loadUrl(url);
              //调用拨号程序
              if (url.startsWith("mailto:") || url.startsWith("geo:")
                  ||url.startsWith("tel:")) {
                Intent intent = new Intent(Intent.ACTION VIEW,
                  Uri.parse(url));
                 startActivity(intent);
                }
              return true;
          }
      });
```

(6) 拦截超链接。

可以使用 WebView 拦截超链接，用 URL 表示拦截到的链接，我们可以对 URL 做判断，比如在线播放音乐的链接，即检测其中是否含有 http://xxx.mp3 这种链接，如果有就调用音乐播放器来播放，或者是在线播放“rtsp://”格式的。例如下面的代码：

```
mWebView.setWebViewClient(new WebView Client(){
/*
此处能拦截超链接的 URL，即拦截 href 请求的内容
*/
public boolean shouldOverrideUrlLoading(WebView  view, String url) {
view.loadUrl(url);
return true;
}
});
```

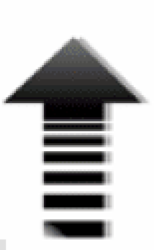
(7) 处理 SslError。

在 Android 中，WebView 是用来加载 HTTP 和 HTTPS 网页到本地应用的控件。在默认情况下，通过 loadUrl(String url)方法，可以顺利加载诸如 http://www.baidu.com 之类的页面。但是，当加载有 SSL 层的 HTTPS 页面(例如 https://money.183.com.cn/)时，如果这个网站的安全证书在 Android 无法得到认证，WebView 就会变成一个空白页，而并不会像 PC 浏览器中那样跳出一个风险提示框。因此，我们必须针对这种情况进行处理。在 Android 处理时需要用到以下两个类。

- import android.NET.http.SslError
- import android.webkit.SslErrorHandler

具体的用法如下：

```
WebView wv = (WebView) findViewById(R.id.webview);
wv.setWebViewClient(new WebViewClient(){
public void onReceivedSslError(WebView view, SslErrorHandler handler,
SslError error){
```




```
//handler.cancel(); 默认的处理方式，WebView 变成空白页
//handler.process();接收证书
//handleMessage(Message msg); 其他处理
}
```

(8) 删除缓存。

可以使用 WebView 删除手机上的缓存。例如下面的代码：

```
private int clearCacheFolder(File dir, long numDays) {
    int deletedFiles = 0;
    if (dir!= null && dir.isDirectory()) {
        try {
            for (File child:dir.listFiles()) {
                if (child.isDirectory()) {
                    deletedFiles += clearCacheFolder(child, numDays);
                }
                if (child.lastModified() < numDays) {
                    if (child.delete()) {
                        deletedFiles++;
                    }
                }
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
    return deletedFiles;
}
```

优先使用缓存的设置代码如下：

```
WebView.getSettings().setCacheMode(WebSettings.LOAD_CACHE_ELSE_NETWORK);
```

使用缓存的设置代码如下。

```
WebView.getSettings().setCacheMode(WebSettings.LOAD_NO_CACHE);
```

(9) 使用 WebView 设置 URL 的加载。

当在 WebView 里打开一个链接时，默认会通过 ActivityManager 寻找合适的浏览器进行打开，如果想避免这种事情的发生的话，可以通过如下流程解决。

① 添加权限。

在文件 AndroidManifest.xml 中声明 android.permission.INTERNET 权限，否则会出现“Web page not available”错误。

② 在要显示的 Activity 中生成一个 WebView 组件：

```
WebView webView = new WebView(this);
```

设置 WebView 基本信息：

如果访问的页面中有 JavaScript，则 WebWiew 必须设置支持 JavaScript。

```
webview.getSettings().setJavaScriptEnabled(true);
```

还需要设置触摸焦点起作用：

```
requestFocus();
```

取消滚动条功能：

```
this.setScrollBarStyle(SCROLLBARS_OUTSIDE_OVERLAY);
```

③ 设置 WevView 要显示的网页。

- 互联网用：

```
webView.loadUrl("http://www.google.com");
```

- 本地文件用：

```
webView.loadUrl("file:///android_asset/XX.html");
```

其中本地文件存放在 assets 目录中。

④ 如果用 WebView 点击链接看了很多网页以后，如果不做任何处理，点击系统的 Back 键，整个浏览器会调用 finish()而结束自身，如果希望浏览的网页回退而不是退出浏览器，需要在当前 Activity 中处理并消费掉该 Back 事件，并覆盖 Activity 类的 onKeyDown(int keyCoder,KeyEvent event)方法。

根据上述做法可知，其实就是实现了一个继承于 WebViewClient 的类，例如下面的代码：

```
public class TestClient extends WebViewClient {

    public boolean shouldOverrideUrlLoading(WebView webview, String url){
        Log.d("TestClient", url);
        //TODO:在此添加 URL 处理代码，如果返回 true，则 WebView 不会请求 AcitvityManager
          打开这个 URL
        return false;
    }
}
```

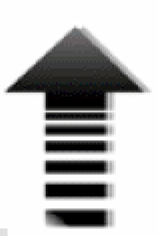
从此以后，就可以通过 WebView.setWebViewClient()设置上述类的实例化对象即可，这也可以算是一种另类的 HTML 页面与 Java 之间的通信手段，甚至可以用在浏览器插件和 Java 程序之间的通信。

6.4.4 使用WebView类浏览网页

实　例	功　能	源码路径
实例 6-1	在手机屏幕中浏览网页	下载路径:\daima\6\wang

在本实例中用到了 Android 系统中的内置 WebKit 引擎，通过此引擎中的 WebView 类来浏览网页。在具体实现时，是通过 WebView.loadUrl 来加载网址的。当从 EditText 中传入要浏览的网址后，可以在 WebView 中加载网页的内容。本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，在里面插入一个 WebView 控件。主要代码如下：



```
<!-- 建立一个 TextView -->
<TextView
android:id="@+id/myTextView1"
android:layout width="fill parent"
android:layout_height="wrap_content"
android:text="@string/hello"
/>
<!-- 建立一个 EditText -->
<EditText
android:id="@+id/myEditText1"
android:layout width="267px"
android:layout height="40px"
android:textSize="18sp"
android:layout x="5px"
android:layout y="32px"
/>
<!-- 建立一个 ImageButton -->
<ImageButton
android:id="@+id/myImageButton1"
android:layout width="wrap content"
android:layout height="wrap content"
android:background="@drawable/white"
android:src="@drawable/go"
android:layout x="275px"
android:layout y="35px"
/>
<!-- 建立一个 WebView -->
<WebView
android:id="@+id/myWebView1"
android:layout height="330px"
android:layout width="300px"
android:layout x="7px"
android:layout y="90px"
android:background="@drawable/black"
android:focusable="false"
/>
```

(2) 编写文件 wang.java，通过 setOnClickListener 监听按钮单击事件，单击网址后面的箭头后会抓取 EditText 中的数据，然后打开此网址，并在 WebView 中显示网页内容。具体代码如下：

```
package irdc.wang;

import irdc.wang.R;
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.webkit.WebView;
import android.widget.EditText;
import android.widget.ImageButton;
```

```
import android.widget.Toast;

 public void onCreate(Bundle savedInstanceState)
 {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   mImageButton1 = (ImageButton)findViewById(R.id.myImageButton1);
   mEditText1 = (EditText)findViewById(R.id.myEditText1);
   mWebView1 = (WebView) findViewById(R.id.myWebView1);

   /*当单击箭头后*/
   mImageButton1.setOnClickListener(new
                               ImageButton.OnClickListener()
   {
     @Override
     public void onClick(View arg0)
     {
       // TODO Auto-generated method stub
       {
         mImageButton1.setImageResource(R.drawable.go_2);
         /*抓取 EditText 中的数据*/
         String strURI = (mEditText1.getText().toString());
         /*  WebView 显示网页内容  */
         mWebView1.loadUrl(strURI);
         Toast.makeText(
             example2.this,getString(R.string.load)+strURI,
                       Toast.LENGTH_LONG)
             .show();
       }
     }
   });
 }
}
```

执行后显示一个文本框，在此可以输入网址，如图 6-4 所示。输入网址并单击后面的▶按钮后，将显示此网页的内容，如图 6-5 所示。

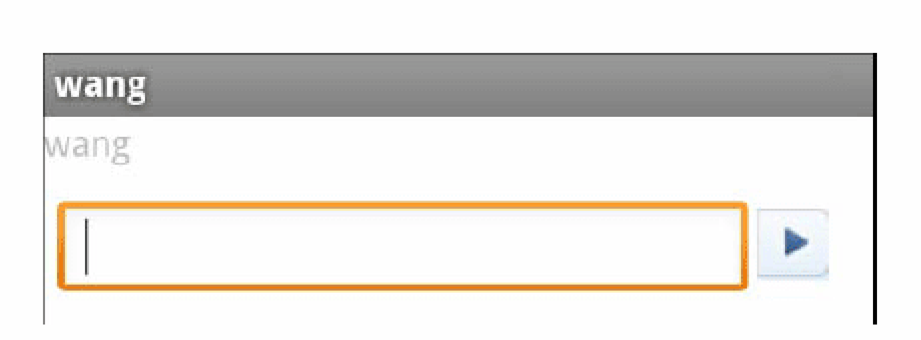


图 6-4　输入网址



图 6-5　打开的网页

6.4.5　使用WebView类加载HTML程序

实　例	功　能	源码路径
实例 6-2	在手机屏幕中加载 HTML 程序	下载路径:\daima\6\HT

HTML 语言是当前主流的网页技术，而 WebView 是一个嵌入式的浏览器，在里面可以直接使用 WebView.loadData()。WebView 将 HTML 标记传递给 WebView 对象，让 Android 手机程序变为 Web 浏览器。这样，网页程序被放在了 WebView 中运行，如同一个 Web Application。

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下：

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:background="@drawable/white"
  android:layout width="fill parent"
  android:layout height="fill parent"
  >
  <!-- 创建一个 TextView -->
  <TextView
  android:id="@+id/myTextView1"
  android:layout width="fill parent"
  android:layout height="wrap content"
  android:textColor="@drawable/blue"
  android:text="@string/hello"
  />
  <!-- 创建一个 WebView -->
  <WebView
  android:id="@+id/myWebView1"
  android:layout height="wrap content"
  android:layout width="wrap content"
  />
</LinearLayout>
```

(2) 编写文件 HT.java，在 loadData 中插入了预先设置好的 HTML 代码，通过 HTML 代码显示了一幅图片和文字，并且实现了超级链接功能。具体代码如下：

```
public class HT extends Activity
{
  private WebView mWebView1;
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mWebView1 = (WebView) findViewById(R.id.myWebView1);
    /*自行设置 WebView 要显示的网页内容*/
    mWebView1.
```

```
    loadData(
    "<html><body><p>aaaaaaa</p>" +
    "<div class='widget-content'> "+
    "<a href=http://www.sohu.com>" +
    "<img src=http://hiphotos.baidu.com/chaojihedan/pic/item/
        bbddf5efc260f133fdfa3cd8.jpg />" +
    "<a href=http://www.sohu.com>Link Blog</a>" +
    "</body></html>", "text/html", "utf-8");
  }
}
```

执行后将显示 HTML 产生的页面，如图 6-6 所示。单击超链接后会来到指定的目标页面。

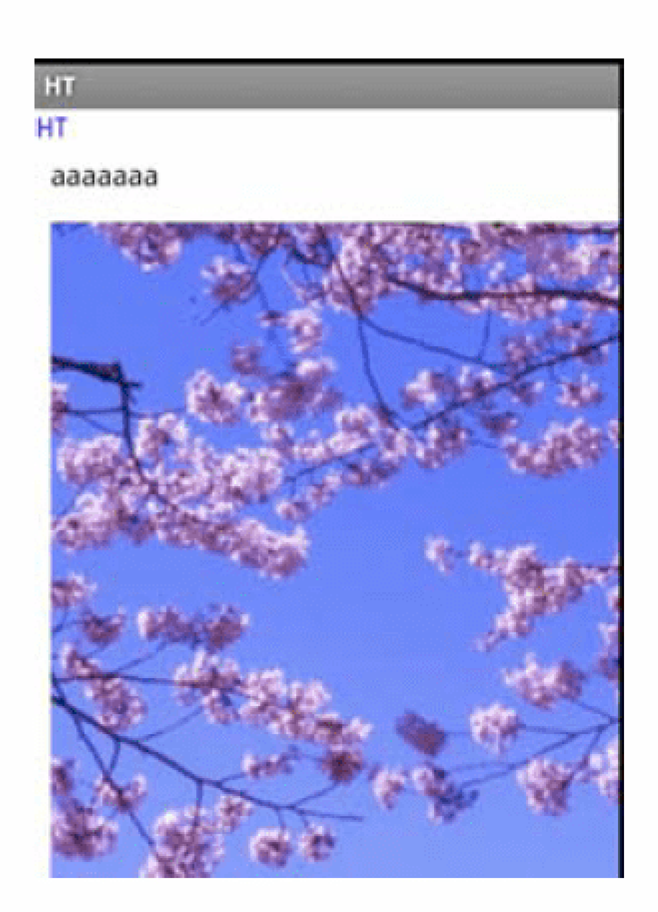


图 6-6 执行效果

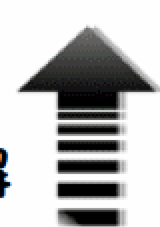
6.4.6 使用WebView加载JavaScript程序

实 例	功 能	源码路径
实例 6-3	使用 WebView 加载 JavaScript 程序	下载路径:\daima\6\RIADemo

在本实例中，预先准备了一个 HTML 文件和一个 JavaScript 文件，本实例的最终目的是在加载 HTML 的同时加载 JavaScript 文件，在 HTML 中显示手机中联系人的信息。本实例的具体实现流程如下。

(1) 准备 HTML 文件 phonebook.html，具体代码如下：

```
<html>
    <head>
        <script type="text/javascript" src="fetchcontacts.js"/>
</head>
<body>
    <div id = "contacts">
        <p> this is a demo </p>
    </div>
</body>
</html>
```



(2) 准备 JavaScript 文件 fetchcontacts.js，具体代码如下：

```
window.onload= function(){
    window.phonebook.debugout("inside js onload");
        //调用 RIAExample.debugout
    var persons = window.phonebook.getContacts();
        //调用 RIAExample.getContacts()
    if(persons){//persons 实际上是 JavaArrayJSWrapper 对象
        window.phonebook.debugout(persons.length() + " of contact
            entries are fetched");
        var contactsE = document.getElementById("contacts");
        var i = 0;
        while(i < persons.length()){
            //persons.length()调用 JavaArrayJSWrapper.length()方法
            pnode = document.createElement("p");
            //persons.get(i)获得 Person 对象
            //然后在 JS 里面直接调用 getName()和 getNumber()获取姓名和号码
           tnode = document.createTextNode("name : " + persons.get(i).getName()
             + " number : " + persons.get(i).getNumber());
            pnode.appendChild(tnode);
            contactsE.appendChild(pnode);
            i ++;
        }
    }else{
        window.phonebook.debugout("persons is undefined");
    }

}
```

(3) 编写布局文件 main.xml，在其中添加一个 WebView 控件。主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
<WebView android:id="@+id/web"
 android:layout width="fill parent" android:layout height="fill parent">
</WebView>
</LinearLayout>
```

(4) 编写文件 Person.java，定义类 Person 来描述一个联系人的信息，它包含联系人姓名和号码。主要代码如下：

```
public class Person {
    String name;
    String phone number;
    public String getName(){

        return name;
    }
```

```
    public String getNumber(){
        return phone number;
    }
}
```

(5) 编写文件 JavaArrayJSWrapper.java，主要代码如下：

```
public class JavaArrayJSWrapper {

    private Object[] innerArray;

    public JavaArrayJSWrapper(Object[] a){
        this.innerArray = a;
    }

    public int length(){
        return this.innerArray.length;
    }

    public Object get(int index){
        return this.innerArray[index];
    }
}
```

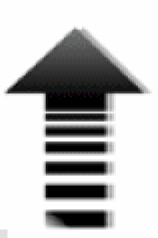
(6) 编写测试文件 RIAExample.java，主要代码如下：

```
package com.example;

import java.util.Vector;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.webkit.WebView;

public class RIAExample extends Activity {
    private WebView web;
    //模拟号码簿
    private Vector<Person> phonebook = new Vector<Person>();
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        this.initContacts();
        web = (WebView)this.findViewById(R.id.web);
        web.getSettings().setJavaScriptEnabled(true);
            //开启 JavaScript 设置，否则 WebView 不执行 JS 脚本
        web.addJavascriptInterface(this, "phonebook");
            //把 RIAExample 的一个实例添加到 JS 的全局对象 window 中，
             //这样就可以使用 window.phonebook 来调用它的方法
        web.loadUrl("file:///android_asset/phonebook.html");//加载网页
```



```
    }

    /**
     * 该方法将在 JS 脚本中，通过 window.phonebook.getContacts()进行调用
     * 返回的 JavaArrayJSWrapper 对象可以使得在 JS 中访问 Java 数组
     * @return
     */
    public JavaArrayJSWrapper getContacts(){
     System.out.println("fetching contacts data");
     Person[] a = new Person[this.phonebook.size()];
     a = this.phonebook.toArray(a);
     return new JavaArrayJSWrapper(a);
    }
    /**
     * 初始化电话号码簿
     */
    public void initContacts(){
     Person p = new Person();
     p.name = "Perter";
     p.phone number = "8888888";
     phonebook.add(p);
     p = new Person();
     p.name = "wangpengl";
     p.phone number = "13000000";
     phonebook.add(p);
    }
    /**
     * 通过 window.phonebook.debugout 来输出 JS 调试信息
     * @param info
     */
    public void debugout(String info){
     Log.i("ss",info);
     System.out.println(info);
    }
}
```

执行后的效果如图 6-7 所示。

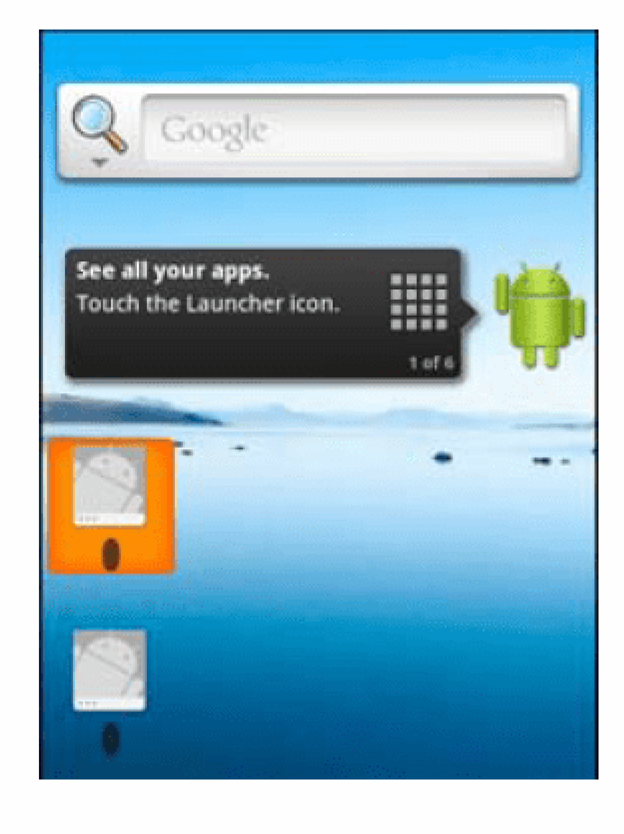


图 6-7　执行效果

本实例的目的是为了说明通过 WebView.addJavascriptInterface 方法可以扩展 JavaScript 的 API，这样可以获取 Android 的数据。由此可见，我们可以使用 Dojo、jQuery 和 Protory 等 JS 框架来搭建 Android 应用程序，来实现更加绚丽的效果。

6.4.7 使用WebView的注意事项

基于 WebView 在 Android 浏览器领域的重要性，下面将简单讲解在使用 WebView 时的注意事项，帮助广大读者，特别是初学者来避免一些不必要的错误和麻烦。

(1) 在文件 AndroidManifest.xml 中必须使用许可 android.permission.INTERNET，否则会出现“Web page not available”的错误。

(2) 如果访问的页面中有 JavaScript，则 WebView 必须设置支持 JavaScript。

```
webview.getSettings().setJavaScriptEnabled(true);
```

(3) 当页面中有链接，如果希望点击链接继续在当前 Browser 中响应，而不是在新开的 Android 系统 Browser 中响应该链接，必须覆盖 WebView 的 WebViewClient 对象，例如下面的代码：

```
mWebView.setWebViewClient(new WebViewClient(){
                public boolean shouldOverrideUrlLoading(WebView view,
                    String url) {
                    view.loadUrl(url);
                    return true;
                }
       });
```

(4) 如果不做任何处理在浏览网页时，点击系统的 Back 键，整个 Browser 会调用 finish()而结束自身，如果希望浏览的网页是回退而不是退出浏览器，需要在当前 Activity 中处理并消耗掉该 Back 事件。例如下面的代码：

```
  public boolean onKeyDown(int keyCode, KeyEvent event) {
        if ((keyCode == KeyEvent.KEYCODE_BACK) && mWebView.canGoBack())
{
            mWebView.goBack();
                  return true;
        }
        return super.onKeyDown(keyCode, event);
    }
```

Android

第 7 章

在Android中开发蓝牙应用

蓝牙是通信领域中新兴的一个专有名词，是一种支持设备短距离(一般 10m 内)通信的无线电技术。它能在包括移动电话、PDA、无线耳机、笔记本电脑、相关外设等众多设备之间进行无线信息交换。本章将简要介绍在 Android 平台中开发蓝牙相关应用的基本知识。

7.1 蓝牙系统的结构

蓝牙系统比较复杂，但是又很常用，要想完全掌握蓝牙应用开发技术，我们需要从底层做起，首先了解它的底层结构。本节将简要讲解蓝牙系统底层结构的基本知识。

7.1.1 蓝牙概述

利用蓝牙技术，能够有效地简化移动通信终端设备之间的通信，也能够成功地简化设备与 Internet 之间的通信，从而使数据传输变得更加迅速高效，为无线通信拓宽道路。蓝牙采用分散式网络结构以及快跳频和短包技术，支持点对点及点对多点通信；工作在全球通用的 2.4GHz ISM(即工业、科学、医学)频段；数据速率为 1Mb/s；采用时分双工传输方案实现全双工传输。

1. 蓝牙的发展历程

蓝牙这个名称来自于第十世纪的一位丹麦国王 Harald Blatand，Blatand 在英文里的意思可以被解释为 Bluetooth(蓝牙)，因为国王喜欢吃蓝莓，牙龈每天都是蓝色的，所以叫蓝牙。

蓝牙的创始人是瑞典爱立信公司，爱立信早在 1994 年就已进行研发。1997 年，爱立信与其他设备生产商联系，并激发了他们对该项技术的浓厚兴趣。1998 年 2 月，5 个跨国大公司，包括爱立信、诺基亚、IBM、东芝及 Intel 组成了一个特殊兴趣小组(SIG)，他们共同的目标是建立一个全球性的小范围无线通信技术，即现在的蓝牙。

蓝牙无线技术规格供全球的成员公司免费使用。许多行业的制造商都积极地在其产品中实施此技术，以减少使用零乱的电线，实现无缝连接、流传输立体声，传输数据可进行语音通信。蓝牙技术在 2.4 GHz 波段运行，该波段是一种无须申请许可证的工业、科技、医学(ISM)无线电波段。正因如此，使用蓝牙技术不需要支付任何费用。但必须向手机提供商注册使用 GSM 或 CDMA，除了设备费用外，不需要为使用蓝牙技术再支付任何费用。

蓝牙技术得到了空前广泛的应用，集成该技术的产品从手机、汽车到医疗设备，使用该技术的用户从消费者、工业市场到企业，等等，不一而足。低功耗、小体积以及低成本的芯片解决方案使得蓝牙技术甚至可以应用于极微小的设备中。

2. 蓝牙的特点

蓝牙技术是一项即时技术，它不要求固定的基础设施，且易于安装和设置。无须电缆即可实现连接。新用户使用亦不费力，我们只需拥有蓝牙品牌产品，检查可用的配置文件，将其连接至使用同一配置文件的另一蓝牙设备即可。后续的 PIN 码流程就如同您在 ATM 机器上操作一样简单。外出时，您可以随身带上您的个人局域网(PAN)，甚至可以与其他网络连接。



3. Android中的蓝牙

Android 包含了对蓝牙网络协议栈的支持，这使得蓝牙设备能够无线连接其他蓝牙设备交换数据。Android 的应用程序框架提供了访问蓝牙功能的 APIs，这些 APIs 让应用程序能够无线连接其他蓝牙设备，实现点对点或点对多点的无线交互功能。

使用蓝牙 APIs，Android 应用程序能够实现以下功能。

- 扫描其他蓝牙设备。
- 查询本地蓝牙适配器(Local Bluetooth Adapter)用于配对蓝牙设备。
- 建立 RFCOMM 信道(channels)。
- 通过服务发现(Service Discovery)连接其他设备。
- 数据通信。
- 管理多个连接。

7.1.2　蓝牙层次结构

Android 平台的蓝牙系统是基于 BlueZ 实现的，是通过 NUX 中一套完整的蓝牙协议栈开源实现的。当前 BlueZ 被广泛应用于各种 Linux 版本中，并被芯片公司移植到各种芯片平台上来使用。在 Linux 2.6 内核中已经包含了完整的 BlueZ 协议栈，在 Android 系统中也已移植并嵌入了 BlueZ 的用户空间实现，并且随着硬件技术的发展而不断更新。

蓝牙技术实际上是一种短距离无线电技术。在 Android 系统中的蓝牙除了使用 Kernel 支持外，还需要用户空间的 BlueZ 的支持。

Android 平台中蓝牙系统的基本层次结构如图 7-1 所示。

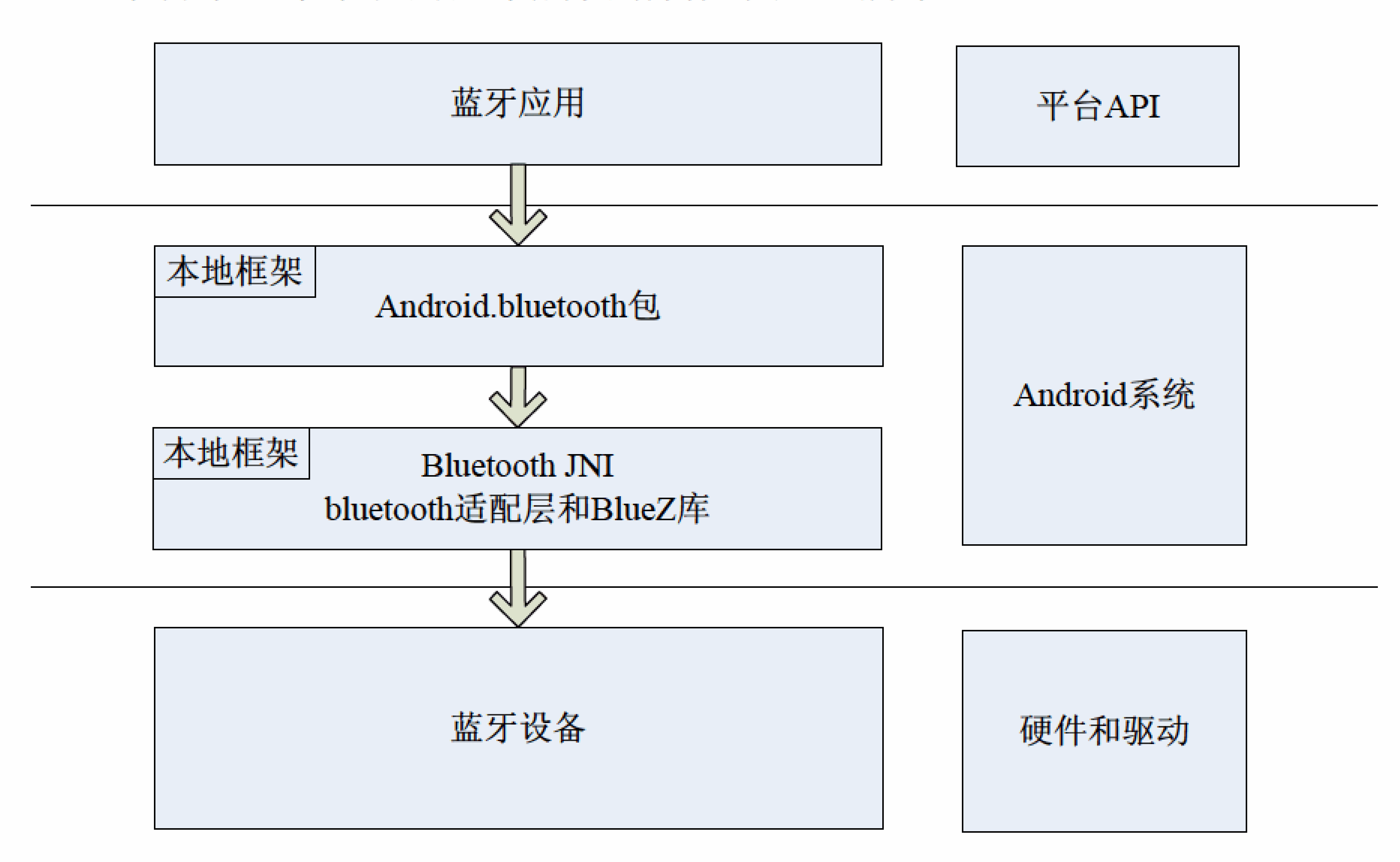


图 7-1　蓝牙系统的层次结构

Android 平台中蓝牙系统从上到下主要包括 Java 框架中的 BlueTooth 类、Android 适配库、BlueZ 库、驱动程序和协议，这几部分的系统结构如图 7-2 所示。

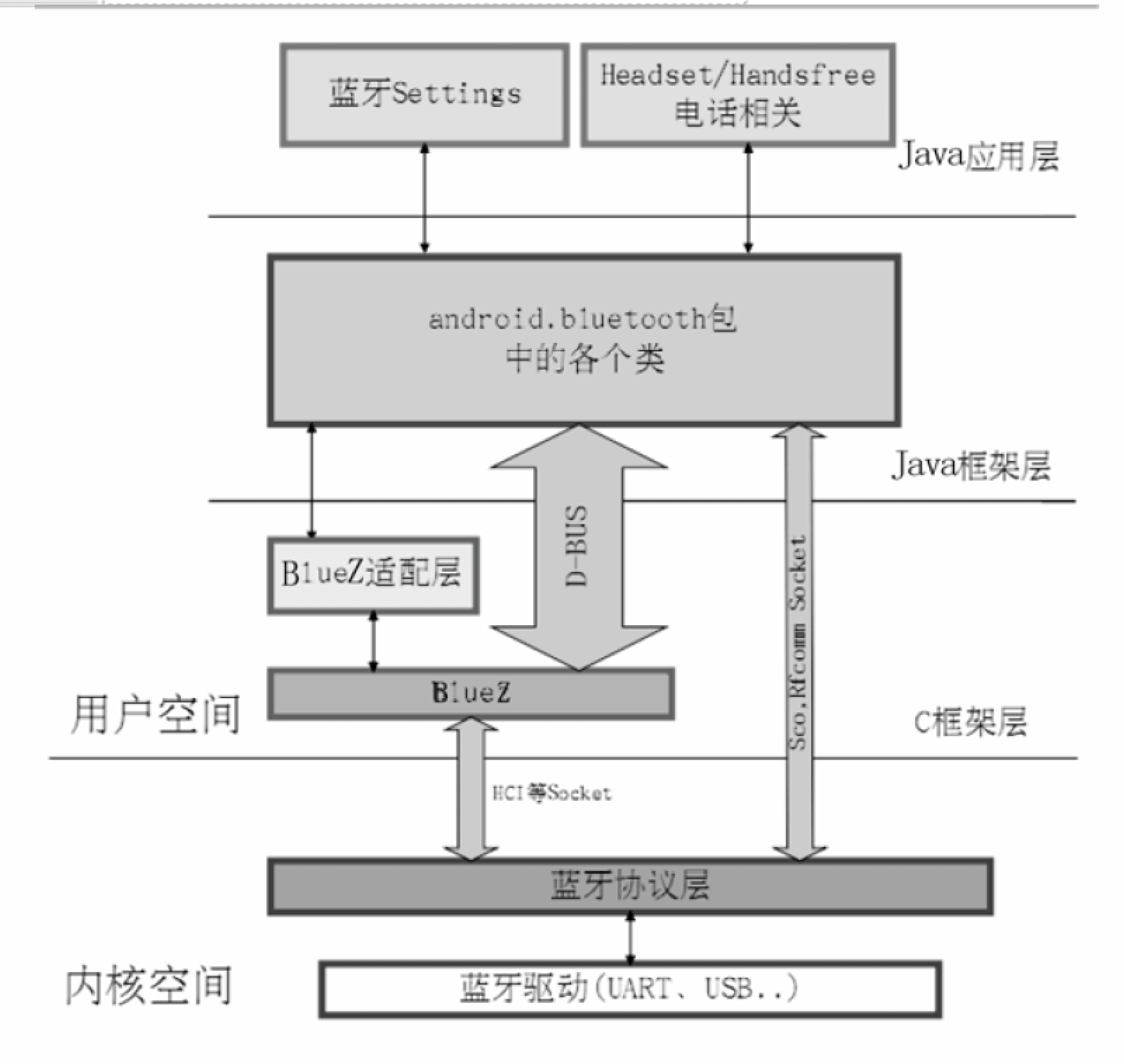


图 7-2　蓝牙系统结构

在图 7-2 中各个层次结构的具体说明如下。

1. BlueZ库

Android 蓝牙设备管理库的路径如下：

```
external/bluez/
```

可以分别生成 libbluetooth.so、libbluedroid.so 和 hcidump 等众多相关工具和库。BlueZ 库提供了对用户空间蓝牙的支持，其中包含了主机控制协议 HCI 以及其他众多内核实现协议的接口，并且实现了所有蓝牙应用模式 Profile。

2. 蓝牙的 JNI部分

此部分的代码路径如下：

```
frameworks/base/core/jni/
```

3. Java框架层

Java 框架层的实现代码保存在如下路径：

```
frameworks/base/core/java/android/bluetooth      //蓝牙部分对应应用程序的 API
frameworks/base/core/java/android/Server         //蓝牙的服务部分
```

蓝牙的服务部分负责管理并使用底层本地服务，并封装成系统服务。而在 android.bluetooth 部分中包含了各个蓝牙平台的 API 部分，以供应用程序层所使用。





4. BlueTooth适配库

BlueTooth 适配库的代码路径如下：

```
system/bluetooth/
```

在此层用于生成库 libbluedroid.so 以及相关工具和库，能够实现对蓝牙设备的管理，例如蓝牙设备的电源管理。

7.1.3　蓝牙在Android和Linux中的差异

在 Android 平台中，蓝牙系统的本地层和框架层都是标准的程序，仅有的区别是蓝牙驱动程序。蓝牙驱动程序包括针对硬件接口的 USB、SDIO 和 UART 驱动，此部分驱动的内容也是标准的。如果使用 UART 蓝牙芯片，则需要使用芯片特定的高速串口。另外在驱动中还包含了电源管理和芯片配置。因为通常硬件接口都比较标准，所以有很多蓝牙芯片通过用户空间的初始化代码直接对芯片进行写入操作，以完成初始化操作。

1. BlueZ

Android 所采用的蓝牙用库空间的库是 BlueZ。它是一套 Linux 平台的蓝牙协议栈完整开源实现，广泛应用在各 Linux 发行版，并被移植到众多移动平台上。在 Android 系统中，BlueZ 提供了很多分散的应用，例如守护进程和一些工具。BlueZ 通过 D-BUS IPC 机制来提供应用层接口。

2. 适配层

BlueZ 的适配层 BlueZ 在 Android 中使用，需要经过 Android 的 BlueZ 适配层的封装。BlueZ 适配层源代码及头文件路径如下：

```
system/bluetooth/
```

该目录中除了包含生成适配层库 libbluedroid.so 的源码之外，还包含了 BlueZ 头文件和 BlueZ 配置文件等目录。由于 BlueZ 使用 D-BUS 作为与上层沟通的接口，适配层构造比较简单，封装了蓝牙的开关功能，以及射频开关。

3. JNI和Java部分

在 Android 中还定义了 BlueTooth 通过 JNI 到上层的接口，此功能保存在如下目录中。

```
frameworks/base/core/jni/
```

此目录中的主要实现文件如下。

- android_bluetooth_BluetoothAudioGateway.cpp
- android_bluetooth_common.cpp
- android_bluetooth_Database.cpp
- android_bluetooth_ScoSocket.cpp
- android_bluetooth_RfcommSocket.cpp
- android_bluetooth_HeadsetBase.cpp

7.2 分析蓝牙源码

要想掌握蓝牙的开发原理，则需要分析 Android 中的蓝牙源码并了解其核心构造，这样才能对蓝牙应用开发做到游刃有余。本节将简要介绍开源 Android 中和蓝牙相关的代码。

7.2.1 初始化蓝牙芯片

初始化蓝牙芯片是通过 BlueZ 工具 hciattach 进行的，此工具在以下目录的文件中实现。

```
external/bluetooth/tools
```

hciattach 命令主要用来初始化蓝牙设备，其命令格式如下：

```
hciattach [-n] [-p] [-b] [-t timeout] [-s initial speed] <tty> <type |
id> [speed] [flow|noflow] [bdaddr]
```

其中最重要的参数是 type 和 speed，type 决定了要初始化的设备的型号，可以使用 hciattach -l 来列出所支持的设备型号。

并不是所有的参数对所有的设备都是适用的，有些设备会忽略一些参数设置，例如，查看 hciattach 的代码就可以看到，多数设备都忽略 bdaddr 参数。hciattach 命令内部的工作步骤是：首先打开制定的 tty 设备，然后做一些通用的设置，如 flow 等，接着设置波特率为 initial_speed，再根据 type 调用各自的初始化代码，最后将波特率重新设置为 speed。所以调用 hciattach 时，要根据自己的实际情况，设置好 initial_speed 和 speed。

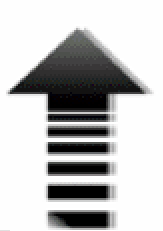
对于 type BCSP 来说，它的初始化代码只做了一件事，就是完成 BCSP 协议的同步操作，它并不对蓝牙芯片做任何的 pskey 的设置。

7.2.2 蓝牙服务

在蓝牙服务方面一般不用我们自己定义，只需要使用初始化脚本文件 init.rc 中的默认内容即可。例如下面的代码：

```
service bluetoothd /system/bin/logwrapper /system/bin/bluetoothd -d -n
    socket bluetooth stream 660 bluetooth bluetooth
    socket dbus bluetooth stream 660 bluetooth bluetooth
    # init.rc does not yet support applying capabilities, so run as root and
    # let bluetoothd drop uid to bluetooth with the right linux capabilities
    group bluetooth net bt admin misc
    disabled

# baudrate change 115200 to 1152000(Bluetooth)
service changebaudrate /system/bin/logwrapper /system/xbin/bccmd 115200
-t bcsp -d /dev/s3c2410_serial1 psset -r 0x1be 0x126e
```



```
    user bluetooth
    group bluetooth net bt admin
    disabled
    oneshot

#service hciattach /system/bin/logwrapper /system/bin/hciattach -n -s
1152000 /dev/s3c2410 serial1 bcsp 1152000
service hciattach /system/bin/logwrapper /system/bin/hciattach -n -s
115200 /dev/s3c2410 serial1 bcsp 115200
    user bluetooth
    group bluetooth net bt admin misc
    disabled

service hfag /system/bin/sdptool add --channel=10 HFAG
    user bluetooth
    group bluetooth net bt admin
    disabled
    oneshot

service hsag /system/bin/sdptool add --channel=11 HSAG
    user bluetooth
    group bluetooth net bt admin
    disabled
    oneshot

service opush /system/bin/sdptool add --channel=12 OPUSH
    user bluetooth
    group bluetooth net bt admin
    disabled
    oneshot

service pbap /system/bin/sdptool add --channel=19 PBAP
    user bluetooth
    group bluetooth net bt admin
    disabled
    oneshot
```

在上述代码中，每一个 Service 后面列出了一种 Android 服务。

7.2.3　管理蓝牙电源

在 Android 系统的目录中实现了 libbluedroid。

```
system/bluetooth/
```

我们可以调用 rfkill 接口来控制电源管理。如果已经实现了 rfkill 接口，则无须再进行配置。如果在文件 init.rc 中已经实现了 hciattach 服务，则说明在 libbluedroid 中已经实现对其调用以操作蓝牙的初始化。

7.3 和蓝牙相关的类

经过本章前面内容的学习，已经了解了 Android 系统中蓝牙的基本知识。根据对上述从底层到应用的学习，了解了蓝牙的工作原理和机制。本节将详细讲解在 Android 系统中和蓝牙相关的类，为读者步入本书后面知识的学习打好基础。

7.3.1 BluetoothSocket类

1. BluetoothSocket类基础

类 BluetoothSocket 的格式如下：

```
public static class BluetoothSocket implements Closeable
```

类 BluetoothSocket 的结构如下：

```
java.lang.Object
android.android.bluetoothSocket
android.widget.android.bluetoothSocket
```

Android 的蓝牙系统和 Socket 套接字密切相关，蓝牙端的监听接口和 TCP 的端口类似，都是使用了 Socket 和 ServerSocket 类。在服务器端，使用 BluetoothServerSocket 类来创建一个监听服务端口。当一个连接被 BluetoothServerSocket 所接受，它会返回一个新的 BluetoothSocket 来管理该连接。在客户端，使用一个单独的 BluetoothSocket 类去初始化一个外接连接和管理该连接。

最常使用的蓝牙端口是 RFCOMM，它是被 Android API 支持的类型。RFCOMM 是一个面向连接、通过蓝牙模块进行的数据流传输方式，它也被称为串行端口规范(Serial Port Profile，SPP)。

为了创建一个 BluetoothSocket 类去连接到一个已知设备，使用方法 BluetoothDevice.createRfcommSocketToServiceRecord()，然后调用 connect()方法尝试一个面向远程设备的连接。这个调用将被阻塞，直到一个连接已经建立或者该连接失效。

为了创建一个 BluetoothSocket 作为服务端(或者“主机”)，每当该端口连接成功后，无论它初始化为客户端，或者被接受作为服务器端，都通过方法 getInputStream()和 getOutputStream()来打开 IO 流，从而获得各自的 InputStream 和 OutputStream 对象。

BluetoothSocket 类的线程是安全的，因为 close()方法总会马上放弃外界操作并关闭服务器端口。

2. BluetoothSocket类的公共方法

1) public void close()方法

功能：马上关闭该端口并且释放所有相关的资源。在其他线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。



异常：IOException。

2) public void connect()方法

功能：尝试连接到远程设备。该方法将阻塞，直到一个连接建立或者失效。如果该方法没有返回异常值，则该端口现在已经建立。当设备查找正在进行的时候，不可尝试创建对远程蓝牙设备的新连接，因为在蓝牙适配器上，设备查找是一个重量级过程，并且会降低一个设备的连接。可以使用 cancelDiscovery()方法取消一个外界的查询。查询并不是由活动所管理，而是作为一个系统服务来运行，所以即使它不能直接请求一个查询，应用程序也总会调用 cancelDiscovery()方法。用 close()方法可以放弃从另一线程而来的调用。

异常：IOException，表示一个错误，例如连接失败。

3) public InputStream getInputStream()方法

功能：通过连接的端口获得输入数据流，即使该端口未连接，该输入数据流也会返回。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立。

返回值：输入流。

异常：IOException。

4) public OutputStream getOutputStream()方法

功能：通过连接的端口获得输出数据流，即使该端口未连接，该输出数据流也会返回。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立。

返回值：输出流。

异常：IOException。

5) public BluetoothDevice getRemoteDevice()方法

功能：获得该端口正在连接或者已经连接的远程设备。

返回值：远程设备。

7.3.2 BluetoothServerSocket类

1. BluetoothServerSocket类基础

类 BluetoothServerSocket 的格式如下：

```
public final class BluetoothServerSocket extends Object implements Closeable
```

类 BluetoothServerSocket 的结构如下：

```
java.lang.Object
android.bluetooth.BluetoothServerSocket
```

2. BluetoothServerSocket类的公共方法

1) public BluetoothSocketaccept(int timeout)方法

功能：阻塞，直到超时时间内的连接建立。在一个成功建立的连接上返回一个已连接的 BluetoothSocket 类。每当该调用返回的时候，它可以在此调用去接收以后新来的连接。用 close()方法可以放弃从另一线程来的调用。

参数：timeout，表示阻塞超时时间。

返回值：已连接的 BluetoothSocket。

异常：IOException，表示出现错误，比如该调用被放弃或者超时。

2) public void close()方法

功能：马上关闭端口，并释放所有相关的资源。在其他线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。关闭 BluetoothServerSocket 不会关闭接收自 accept()的任意 BluetoothSocket。

异常：IOException。

7.3.3 BluetoothAdapter类

1. BluetoothAdapter类基础

类 BluetoothAdapter 的格式如下：

```
public final class BluetoothAdapter extends Object
```

类 BluetoothAdapter 的结构如下：

```
java.lang.Object
android.bluetooth.BluetoothAdapter
```

BluetoothAdapter 代表本地的蓝牙适配器设备，通过此类可以让用户能执行基本的蓝牙任务。例如初始化设备的搜索、查询可匹配的设备集、使用一个已知的 MAC 地址来初始化一个 BluetoothDevice 类、创建一个 BluetoothServerSocket 类以监听其他设备对本机的连接请求等。

为了得到这个代表本地蓝牙适配器的 BluetoothAdapter 类，需要调用静态方法 getDefaultAdapter()，这是所有蓝牙动作使用的第一步。当拥有本地适配器以后，用户可以获得一系列的 BluetoothDevice 对象，这些对象代表所有拥有 getBondedDevice()方法的已经匹配的设备；用 startDiscovery()方法来开始设备的搜寻；或者创建一个 BluetoothServerSocket 类，通过 listenUsingRfcommWithServiceRecord(String, UUID)方法来监听新来的连接请求。

注意： 大部分方法需要 BLUETOOTH 权限，一些方法同时需要 BLUETOOTH_ADMIN 权限。

2. BluetoothAdapter类的常量

1) String ACTION_DISCOVERY_FINISHED

广播事件：本地蓝牙适配器已经完成设备的搜寻过程。需要 BLUETOOTH 权限接收。

常量值：android.bluetooth.adapter.action.DISCOVERY_FINISHED。

2) String ACTION_DISCOVERY_STARTED

广播事件：本地蓝牙适配器已经开始对远程设备的搜寻过程。它通常涉及一个大概需时 12 秒的查询扫描过程，紧跟着的是一个对每个获取到自身蓝牙名称的新设备的页面扫描。用户会发现一个把 ACTION_FOUND 常量通知为远程蓝牙设备的注册。设备查找是一个重量级过程，当查找正在进行的时候，用户不能尝试对新的远程蓝牙设备进行连接，同时存在的连接将获得有限制的带宽以及高等待时间。用户可用 cancelDiscovery()类来取消



正在执行的查找进程。需要 BLUETOOTH 权限接收。

常量值：android.bluetooth.adapter.action.DISCOVERY_STARTED。

3) String ACTION_LOCAL_NAME_CHANGED

广播活动：本地蓝牙适配器已经更改了它的蓝牙名称。该名称对远程蓝牙设备是可见的，它总是包含了一个带有名称的 EXTRA_LOCAL_NAME 附加域。需要 BLUETOOTH 权限接收。

常量值：android.bluetooth.adapter.action.LOCAL_NAME_CHANGED。

4) String ACTION_REQUEST_DISCOVERABLE

Activity 活动：显示一个请求被搜寻模式的系统活动。如果蓝牙模块当前未打开，该活动也将请求用户打开蓝牙模块。被搜寻模式和 SCAN_MODE_CONNECTABLE_DISCOVERABLE 等价。当远程设备执行查找进程的时候，它允许其发现该蓝牙适配器。从隐私安全考虑，Android 不会将被搜寻模式设置为默认状态。该意图的发送者可以选择性地运用 EXTRA_DISCOVERABLE_DURATION 这个附加域去请求发现设备的持续时间。普遍来说，对于每一请求，默认的持续时间为 120 秒，最大值则可达到 300 秒。

Android 运用 onActivityResult(int, int, Intent)回收方法来传递该活动结果的通知。被搜寻的时间(以秒为单位)将通过 resultCode 值来显示，如果用户拒绝被搜寻，或者设备产生了错误，则通过 RESULT_CANCELED 值来显示。

每当扫描模式变化的时候，应用程序可以通过 ACTION_SCAN_MODE_CHANGED 值来监听全局的消息通知。比如，当设备停止被搜寻以后，该消息可以被系统通知给应用程序。需要 BLUETOOTH 权限。

常量值：android.bluetooth.adapter.action.REQUEST_DISCOVERABLE。

5) String ACTION_REQUEST_ENABLE

Activity 活动：显示一个允许用户打开蓝牙模块的系统活动。当蓝牙模块完成打开工作，或者当用户决定不打开蓝牙模块时，系统活动将返回该值。Android 运用 onActivityResult(int, int, Intent)回收方法来传递该活动结果的通知。如果蓝牙模块被打开，将通过 resultCode 值 RESULT_OK 来显示；如果用户拒绝该请求，或者设备产生了错误，则通过 RESULT_CANCELED 值来显示。每当蓝牙模块被打开或者关闭，应用程序可以通过 ACTION_STATE_CHANGED 值来监听全局的消息通知。需要 BLUETOOTH 权限。

常量值：android.bluetooth.adapter.action.REQUEST_ENABLE。

6) String ACTION_SCAN_MODE_CHANGED

广播活动：指明蓝牙扫描模块或者本地适配器已经发生变化。它总是包含 EXTRA_SCAN_MODE 和 EXTRA_PREVIOUS_SCAN_MODE。这两个附加域各自包含了新的和旧的扫描模式。需要 BLUETOOTH 权限。

常量值：android.bluetooth.adapter.action.SCAN_MODE_CHANGED。

7) String ACTION_STATE_CHANGED

广播活动：指明蓝牙适配器的状态已经改变，例如蓝牙模块已经被打开或者关闭。它总是包含 EXTRA_STATE 和 EXTRA_PREVIOUS_STATE。这两个附加域各自包含了新的和旧的状态。需要 BLUETOOTH 权限接收。

常量值：android.bluetooth.adapter.action.STATE_CHANGED。

8) int ERROR

功能：标记该类的错误值，确保和该类中的其他整数常量不相等。它为需要一个标记错误值的函数提供了便利。例如：

```
Intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAdapter.ERROR)
```

常量值：-2147483648(0x80000000)。

9) String EXTRA_DISCOVERABLE_DURATION

功能：试图在 ACTION_REQUEST_DISCOVERABLE 常量中作为一个可选的整型附加域，来为短时间内的设备发送请求一个特定的持续时间。默认值为 120 秒，超过 300 秒的请求将被限制。这些值是可以变化的。

常量值：android.bluetooth.adapter.extra.DISCOVERABLE_DURATION。

10) String EXTRA_LOCAL_NAME

功能：试图在 ACTION_LOCAL_NAME_CHANGED 常量中作为一个字符串附加域，来请求本地蓝牙的名称。

常量值：android.bluetooth.adapter.extra.LOCAL_NAME。

11) String EXTRA_PREVIOUS_SCAN_MODE

功能：试图在 ACTION_SCAN_MODE_CHANGED 常量中作为一个整型附加域，来请求以前的扫描模式。可能值如下：

- SCAN_MODE_NONE
- SCAN_MODE_CONNECTABLE
- SCAN_MODE_CONNECTABLE_DISCOVERABLE

常量值：android.bluetooth.adapter.extra.PREVIOUS_SCAN_MODE。

12) String EXTRA_PREVIOUS_STATE

功能：试图在 ACTION_STATE_CHANGED 常量中作为一个整型附加域，来请求以前的供电状态。可能值如下：

- STATE_OFF
- STATE_TURNING_ON
- STATE_ON
- STATE_TURNING_OFF

常量值：android.bluetooth.adapter.extra.PREVIOUS_STATE。

13) String EXTRA_SCAN_MODE

功能：试图在 ACTION_SCAN_MODE_CHANGED 常量中作为一个整型附加域，来请求当前的扫描模式，可以取的值如下：

- SCAN_MODE_NONE
- SCAN_MODE_CONNECTABLE
- SCAN_MODE_CONNECTABLE_DISCOVERABLE

常量值：android.bluetooth.adapter.extra.SCAN_MODE。

14) String EXTRA_STATE

功能：试图在 ACTION_STATE_CHANGED 常量中作为一个整型附加域，来请求当前的供电状态。可以取的值如下：



- STATE_OFF
- STATE_TURNING_ON
- STATE_ON
- STATE_TURNING_OFF

常量值：android.bluetooth.adapter.extra.STATE。

15) int SCAN_MODE_CONNECTABLE

功能：指明在本地蓝牙适配器中，查询扫描功能失效，但页面扫描功能有效。因此该设备不能被远程蓝牙设备发现，但如果以前曾经发现过该设备，则远程设备可以对其进行连接。

常量值：21(0x00000015)。

16) int SCAN_MODE_CONNECTABLE_DISCOVERABLE

功能：指明在本地蓝牙适配器中，查询扫描功能和页面扫描功能都有效。因此该设备既可以被远程蓝牙设备发现，也可以被其连接。

常量值：23 (0x00000017)。

17) int SCAN_MODE_NONE

功能：指明在本地蓝牙适配器中，查询扫描功能和页面扫描功能都失效。因此该设备既不能被远程蓝牙设备发现，也不能被其连接。

常量值：20 (0x00000014)。

18) int STATE_OFF

功能：指明本地蓝牙适配器模块已经关闭。

常量值：10 (0x0000000a)。

19) int STATE_ON

功能：指明本地蓝牙适配器模块已经打开，并且准备被使用。

20) int STATE_TURNING_OFF

功能：指明本地蓝牙适配器模块正在关闭。本地客户端可以立刻尝试友好地断开任意外部连接。

常量值：13 (0x0000000d)。

21) int STATE_TURNING_ON

功能：指明本地蓝牙适配器模块正在打开。然而本地客户在尝试使用这个适配器之前需要为 STATE_ON 状态而等待。

常量值：11 (0x0000000b)。

3. BluetoothAdapter类的公共方法

1) public boolean cancelDiscovery()方法

功能：取消当前的设备发现查找进程，需要 BLUETOOTH_ADMIN 权限。因为对蓝牙适配器而言，查找是一个重量级的过程，因此这个方法必须在尝试连接到远程设备前使用 connect()方法进行调用。发现的过程不会由活动来进行管理，但是它会作为一个系统服务来运行，因此即使它不能直接请求这样的一个查询动作，也必须取消该搜索进程。如果蓝牙状态不是 STATE_ON，这个 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_

CHANGED 更新成 STATE_ON。

返回值：成功则返回 true，有错误则返回 false。

2) public static boolean checkBluetoothAddress (String address)方法

功能：验证皆如“00:43:A8:23:10:F0”之类的蓝牙地址，字母必须为大写才有效。

参数：address，字符串形式的蓝牙模块地址。

返回值：地址正确则返回 true，否则返回 false。

3) public boolean disable()方法

功能：关闭本地蓝牙适配器，不能在没有明确关闭蓝牙的用户动作中使用。这个方法友好地停止所有的蓝牙连接，停止蓝牙系统服务，以及对所有基础蓝牙硬件进行断电。没有用户的直接同意，蓝牙永远不能被禁止。这个 disable()方法只提供了一个应用，该应用包含了一个改变系统设置的用户界面，例如电源控制应用。

这是一个异步调用方法：该方法将马上获得返回值，用户要通过监听 ACTION_STATE_CHANGED 值来获取随后的适配器状态改变的通知。如果该调用返回 true 值，则该适配器状态会立刻从 STATE_ON 转向 STATE_TURNING_OFF，稍后则会转为 STATE_OFF 或者 STATE_ON。如果该调用返回 false，那么说明系统已经有一个保护蓝牙适配器被关闭的问题，比如该适配器已经被关闭了。

需要 BLUETOOTH_ADMIN 权限。

返回值：如果蓝牙适配器的停止进程已经开启则返回 true，如果产生错误则返回 false。

4) public boolean enable()方法

功能：打开本地蓝牙适配器，不能在没有明确打开蓝牙的用户动作中使用。该方法将为基础蓝牙硬件供电，并且启动所有的蓝牙系统服务。没有用户的直接同意，蓝牙永远不能被禁止。如果用户为了创建无线连接而打开了蓝牙模块，则其需要 ACTION_REQUEST_ENABLE 值，该值将提出一个请求用户允许以打开蓝牙模块的会话。这个 enable()值只提供了一个应用，该应用包含了一个改变系统设置的用户界面，例如电源控制应用。

这是一个异步调用方法：该方法将马上获得返回值，用户要通过监听 ACTION_STATE_CHANGED 值来获取随后的适配器状态改变的通知。如果该调用返回 true 值，则该适配器状态会立刻从 STATE_OFF 转向 STATE_TURNING_ON，稍后则会转为 STATE_OFF 或者 STATE_ON。如果该调用返回 false，那么说明系统已经有一个保护蓝牙适配器被打开的问题，比如飞行模式，或者该适配器已经被打开。

需要 BLUETOOTH_ADMIN 权限。

返回值：如果蓝牙适配器的打开进程已经开启则返回 true，如果产生错误则返回 false。

5) public String getAddress()方法

功能：返回本地蓝牙适配器的硬件地址，例如：

```
00:11:22:AA:BB:CC
```

需要 BLUETOOTH 权限。



返回值：字符串形式的蓝牙模块地址。

6) public Set<BluetoothDevice> getBondedDevices()方法

功能：返回已经匹配到本地适配器的 BluetoothDevice 类的对象集合。如果蓝牙状态不是 STATE_ON，这个 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。需要 BLUETOOTH 权限。

返回值：未被修改的 BluetoothDevice 类的对象集合，如果有错误则返回 null。

7) public static synchronized BluetoothAdapter getDefaultAdapter()方法

功能：获取对默认本地蓝牙适配器的操作权限。目前 Android 只支持一个蓝牙适配器，但是 API 可以被扩展为支持多个适配器。该方法总是返回默认的适配器。

返回值：返回默认的本地适配器，如果蓝牙适配器在该硬件平台上不被支持，则返回 null。

8) public String getName()方法

功能：获取本地蓝牙适配器的蓝牙名称，该名称对于外界蓝牙设备而言是可见的。需要 BLUETOOTH 权限。

返回值：该蓝牙适配器名称，如果有错误则返回 null。

9) public BluetoothDevice getRemoteDevice (String address)方法

功能：为给予的蓝牙硬件地址获取一个 BluetoothDevice 对象。合法的蓝牙硬件地址必须为大写，格式类似于“00:11:22:33:AA:BB”。checkBluetoothAddress(String)方法可以用来验证蓝牙地址的正确性。BluetoothDevice 类对于合法的硬件地址总会产生返回值，即使这个适配器从未见过该设备。

参数：address　合法的蓝牙 MAC 地址。

异常：IllegalArgumentException，如果地址不合法。

10) public int getScanMode()方法

功能：获取本地蓝牙适配器的当前蓝牙扫描模式，蓝牙扫描模式决定本地适配器可连接并且/或者可被远程蓝牙设备所连接。需要 BLUETOOTH 权限，可能的取值如下。

- SCAN_MODE_NONE
- SCAN_MODE_CONNECTABLE
- SCAN_MODE_CONNECTABLE_DISCOVERABLE

如果蓝牙状态不是 STATE_ON，则这个 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。

返回值：扫描模式。

11) public int getState()方法

功能：获取本地蓝牙适配器的当前状态，需要 BLUETOOTH 类。可能的取值如下：

- STATE_OFF
- STATE_TURNING_ON
- STATE_ON
- STATE_TURNING_OFF

返回值：蓝牙适配器的当前状态。

12) public boolean isDiscovering()方法

功能：如果当前蓝牙适配器正处于设备发现查找进程中，则返回 true。设备查找是一

个重量级过程，当查找正在进行的时候，用户不能尝试对新的远程蓝牙设备进行连接，同时存在的连接将获得有限制的带宽以及高等待时间。用户可用 cancelDiscovery()类来取消正在执行的查找进程。

应用程序也可以为 ACTION_DISCOVERY_STARTED 或 ACTION_DISCOVERY_FINISHED 进行注册，从而在查找开始或者完成的时候，可以获得通知。

如果蓝牙状态不是 STATE_ON，这个 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。需要 BLUETOOTH 权限。

返回值：如果正在查找，则返回 true。

13) public boolean isEnabled()方法

功能：如果蓝牙正处于打开状态并可用，则返回真值，与 getBluetoothState()==STATE_ON 等价，需要 BLUETOOTH 权限。

返回值：如果本地适配器已经打开，则返回 true。

14) public BluetoothServerSocket listenUsingRfcommWithServiceRecord(String name, UUID uuid)方法

功能：创建一个正在监听的安全的带有服务记录的无线射频通信(RFCOMM)蓝牙端口。一个对该端口进行连接的远程设备将被认证，而对该端口的通信将被加密。使用 accept()方法可以获取从监听 BluetoothServerSocket 处新来的连接。该系统分配一个未被使用的无线射频通信通道来进行监听。

该系统也将注册一个服务探索协议(SDP)记录，该记录带有一个包含了特定的通用唯一识别码(Universally Unique Identifier，UUID)、服务器名称和自动分配通道的本地 SDP 服务。远程蓝牙设备可以用相同的 UUID 来查询自己的 SDP 服务器，并搜寻连接到了哪个通道上。如果该端口已经关闭，或者该应用程序异常退出，则 SDP 记录会被移除。使用 createRfcommSocketToServiceRecord(UUID)可以从另一使用相同 UUID 的设备来连接到这个端口。需要 BLUETOOTH 权限。

参数：

- name：SDP 记录下的服务器名。
- uuid：SDP 记录下的 UUID。

返回值：一个正在监听的无线射频通信蓝牙服务端口。

异常：IOException，表示产生错误，比如蓝牙设备不可用，或者许可无效，或者通道被占用。

15) public boolean setName(String name)方法

功能：设置蓝牙或者本地蓝牙适配器的昵称，这个名字对于外界蓝牙设备而言是可见的。合法的蓝牙名称最多拥有 248 位 UTF-8 字符，但是很多外界设备只能显示前 40 个字符，有些可能只显示前 20 个字符。

如果蓝牙状态不是 STATE_ON，这个 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。需要 BLUETOOTH_ADMIN 权限。

参数：name，一个合法的蓝牙名称。

返回值：如果该名称已被设定，则返回 true，否则返回 false。



16) public boolean startDiscovery()方法

功能：开始对远程设备进行查找的进程，它通常涉及一个大概需时 12 秒的查询扫描过程，紧跟着的是一个对每个获取到自身蓝牙名称的新设备的页面扫描。这是一个异步调用方法，该方法将马上获得返回值，注册 ACTION_DISCOVERY_STARTED and ACTION_DISCOVERY_FINISHED 意图准确地确定该搜索是处于开始阶段还是完成阶段。注册 ACTION_FOUND 以获得远程蓝牙设备已找到的通知。

设备查找是一个重量级过程。当查找正在进行的时候，用户不能尝试对新的远程蓝牙设备进行连接，同时存在的连接将获得有限制的带宽以及高等待时间。用户可用 cancelDiscovery()类来取消正在执行的查找进程。发现的过程不会由活动来进行管理，但是它会作为一个系统服务来运行，因此即使它不能直接请求这样的一个查询动作，也必须取消该搜索进程。设备搜寻只寻找已经被连接的远程设备。许多蓝牙设备默认不会被搜寻到，并且需要进入到一个特殊的模式当中。

如果蓝牙状态不是 STATE_ON，这个 API 将返回 false。蓝牙打开后，等待 ACTION_STATE_CHANGED 更新成 STATE_ON。需要 BLUETOOTH_ADMIN 权限。

返回值：成功返回 true，错误返回 false。

7.3.4　BluetoothClass.Service类

类 BluetoothClass.Service 的格式如下：

```
public static final class BluetoothClass.Service extends Object
```

类 BluetoothClass.Service 的结构如下：

```
java.lang.Object
android.bluetooth.BluetoothClass.Service
```

BluetoothClass.Service 类用于定义所有的服务类常量，任意 BluetoothClass 由 0 个或多个服务类编码组成。类 BluetoothClass.Service 中包含如下常量。

- int AUDIO
- int CAPTURE
- int INFORMATION
- int LIMITED_DISCOVERABILITY
- int NETWORKING
- int OBJECT_TRANSFER
- int POSITIONING
- int RENDER
- int TELEPHONY

7.3.5　BluetoothClass.Device.Major类

类 BluetoothClass.Device.Major 的格式如下：

```
public static class BluetoothClass.Device.Major extends Object
```

类 BluetoothClass.Device.Major 的结构如下：

```
java.lang.Object
android.bluetooth.BluetoothClass.Device.Major
```

类 BluetoothClass.Device.Major 用于定义所有的主要设备类常量，各个常量如下。

- int AUDIO_VIDEO
- int COMPUTER
- int HEALTH
- int IMAGING
- int MISC
- int NETWORKING
- int PERIPHERAL
- int PHONE
- int TOY
- int UNCATEGORIZED
- int WEARABLE

7.3.6 BluetoothClass.Device类

类 BluetoothClass.Device 的格式如下：

```
public final class BluetoothClass.Device extends Object
```

类 BluetoothClass.Device 的结构如下：

```
java.lang.Object
android.bluetooth.BluetoothClass.Device
```

类 BluetoothClass.Device 用于定义所有的设备类的常量，每个 BluetoothClass 有一个带有主要和较小部分的设备类进行编码。其中的常量代表主要和较小的设备类部分(完整的设备类)的组合。

BluetoothClass.Device 有一个内部类，此内部类定义了所有的主要设备类常量。内部类的定义格式如下：

```
class BluetoothClass.Device
```

7.3.7 BluetoothClass类

1. BluetoothClass类基础

类 BluetoothClass 的格式如下：

```
public final class BluetoothClass extends Object implements Parcelable
```

类 BluetoothClass 的结构如下：

```
java.lang.Object
```



```
android.bluetooth.BluetoothClass
```

类 BluetoothClass 代表一个描述了设备通用特性和功能的蓝牙类。比如一个蓝牙类会指定如电话、计算机或耳机的通用设备类型，可以提供如音频或者电话的服务。每个蓝牙类都是由 0 个或更多的服务类，以及一个设备类组成。设备类将被分解成主要和较小的设备类部分。

BluetoothClass 用作能粗略描述一个设备(比如关闭用户界面上一个图标的设备)的搜索，但当蓝牙服务事实上是被一个设备所支撑的时候，BluetoothClass 的描述则不那么可信任。精确的服务搜寻通过 SDP 请求来完成。当运用 createRfcommSocketToServiceRecord(UUID)和 listenUsingRfcommWithServiceRecord(String, UUID)来创建 RFCOMM 端口的时候，SDP 请求就会自动执行。

使用 getBluetoothClass()方法可以获取为远程设备所提供的类。

2. BluetoothClass类的内部类

BluetoothClass 类有以下两个内部类。

- class BluetoothClass.Device：用于定义所有设备类的常量。
- class BluetoothClass.Service：用于定义所有服务类的常量。

3. BluetoothClass类的公共方法

1) public int describeContents()方法

功能：描述包含在可封装编组的表示中所有特殊对象的种类。

返回值：一个指示被 Parcelabel 所排列的特殊对象类型集合的位掩码。

2) public boolean equals(Object o)方法

功能：比较带有特定目标的常量，如果相等则标示出来。为了保证其相等，o 必须代表相同的对象，该对象作为这个使用类依赖比较的常量。通常约定，该比较既要可移植又要灵活。当且仅当 o 是一个作为接收器(使用==操作符来做比较)的精确相同的对象时，这个对象的实现才返回 true 值。子类通常实现 equals(Object)方法，这样它才会重视这两个对象的类型和状态。

在 Android 中约定，对于 equals(Object)和 hashCode()方法，如果 equals 对于任意两个对象返回真值，那么 hashCode()必须对这些对象返回相同的值。这意味着对象的子类通常都覆盖或者都不覆盖这两个方法。

参数：需要对比常量的对象。

返回值：如果特定的对象和该对象相等则返回 true，否则返回 false。

3) public int getDeviceClass()方法

功能：返回 BluetoothClass 中的设备类部分(主要的和较小的)，从函数中返回的值可以和在 BluetoothClass.Device 中的公共常量做比较，从而确定哪个设备类在这个蓝牙类中是被编码的。

返回值：设备类部分。

4) public int getMajorDeviceClass()方法

功能：返回 BluetoothClass 中设备类的主要部分，从函数中返回的值可以和在 BluetoothClass.Device.Major 中的公共常量作比较，从而确定哪个主要类在这个蓝牙类中是

被编码的。

返回值：主要设备类部分。

5) public boolean hasService(int service)方法

功能：如果该指定服务类被 BluetoothClass 所支持，则返回 true。在 BluetoothClass.Service 中，合法的服务类是公共常量，比如 AUDIO 类。

参数：Service 合法服务类。

返回值：如果该服务类可被支持，则返回 true。

6) public int hashCode()方法

功能：返回这个对象的整型哈希码。按约定，任意两个在 equals(Object)中返回 true 的对象必须返回相同的哈希码。这意味着对象的子类通常都覆盖或者都不覆盖这两个方法。除非同等对比信息发生改变，否则哈希码不随时间改变而改变。

返回值：该对象的哈希码。

7) public String toString()方法

功能：返回这个对象的字符串，该字符串包含精确且可读的描述。系统鼓励子类去重写该方法，并且提供了能对该对象的类型和数据进行重构的实现方法。默认的实现方法只是简单地把类名、“@”符号和该对象 hashCode()方法的十六进制数连接起来。

返回值：该对象中一个可被打印的字符串。

8) public void writeToParcel(Parcel out, int flags)方法

功能：将类的数据写入外部提供的 Parcel 中。

参数：

- out：对象需要被写入的 Parcel。
- flags：对象需要如何被写入有关的附加标志。可能是 0，或者可能是 PARCELABLE_WRITE_RETURN_VALUE。

注意： 到此为止，Android 中的蓝牙类介绍完毕。我们在调用这些类时要注意，首先确保 API Level 至少为版本 5 以上，并且还需注意添加相应的权限，比如在使用通信时需要在文件 androidmanifest.xml 中加入<uses-permission android:name="android.permission.BLUETOOTH" />权限，而在开关蓝牙时需要加入 android.permission.BLUETOOTH_ADMIN 权限。

7.4 Android 蓝牙的基本应用

经过前面内容的学习，已经了解了 Android 系统中和蓝牙功能相关的类。本节将简要介绍 Android 蓝牙开发中常见应用功能的基本知识，为读者步入本书后面内容的学习打下基础。

7.4.1 使用BluetoothAdapter类

通过使用 BluetoothAdapter 类，我们可以在 Android 设备上查找周边的蓝牙设备，然



后配对或绑定。蓝牙通信是基于唯一地址 MAC 来相互传输的，考虑到安全问题，蓝牙通信时需要先配对，然后开始相互连接，连接后设备将会共享同一个 RFCOMM 通道以便相互传输数据。

1. 查找发现蓝牙设备

在 Android 中查找发现蓝牙设备时，使用 BluetoothAdapter 类的 startDiscovery()方法即可执行一个异步方式获取周边的蓝牙设备，因为这是一个异步的方法，所以我们不需要考虑线程被阻塞问题，整个过程大约需要 12 秒。这时需要紧接着注册一个 BroadcastReceiver 对象来接收查找到的蓝牙设备信息，通过过滤 ACTION_FOUND Intent 动作可以获取每个远程设备的详细信息，也就是在 Intent 字段通过附加参数 EXTRA_DEVICE 和 EXTRA_CLASS 来获取，在其中包含了每个 BluetoothDevice 对象和对象的该设备类型 BluetoothClass。下面是查找发现蓝牙设备的演示代码。

```
private final BroadcastReceiver cwjReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
           if (BluetoothDevice.ACTION_FOUND.equals(action)) {
             BluetoothDevice device = intent.getParcelableExtra
                (BluetoothDevice.EXTRA_DEVICE);
             myArrayAdapter.add(device.getName() + " android123 " +
                device.getAddress()); //获取设备名称和 MAC 地址
        }
    }
};
// 注册这个 BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(cwjReceiver, filter);
```

在这个过程中读者需要注意，务必在 Service 或 Activity 中重写 onDestory()方法，使用 unregisterReceiver 方法来注册 BroadcastReceiver 对象以保证资源被正确回收。

2. 配对绑定蓝牙设备

在 Android 系统配对一个蓝牙设备时，可以调用 BluetoothAdapter 类的 getBondedDevices()方法来获取已经配对的设备，该方法将会返回一个 BluetoothDevice 数组来区分每个已经配对的设备。例如下面的演示代码。

```
Set<BluetoothDevice> pairedDevices =
cwjBluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0)  //如果获取的结果大于 0，则开始逐个解析
 {
    for (BluetoothDevice device : pairedDevices) {
         //获取每个设备的名称和 MAC 地址添加到数组适配器 myArrayAdapter 中
        myArrayAdapter.add(device.getName() + " android123 " +
          device.getAddress());
    }
}
```

要想让别人的蓝牙设备发现自己的设备，则需要设置自己的机器。要用户确认操作，

不需要获取底层蓝牙服务实例，只需通过一个 Intent 来传递 ACTION_REQUEST_DISCOVERABLE 参数即可。这里通过 startActivityForResult 来强制获取一个结果，重写 startActivityForResult()方法获取执行结果。返回结果有 RESULT_OK 和 RESULT_CANCELLED，分别代表开启和取消(失败)，当然最简单的方法是直接执行。下面是演示代码。

```
Intent cwjIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
cwjIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(cwjIntent);
```

设置完成后，系统会弹出一个询问用户是否允许的对话框。

3. 建立通信

建立一个蓝牙通信必须经过四个步骤，分别是获取本地蓝牙设备、查找远程设备、配对(已配对设备将会忽略这步的细节)、连接设备和传输数据。

在 Android 平台中需要先查找本地活动的蓝牙适配器，通过 BluetoothAdapter 类的 getDefaultAdapter()方法获得一个系统默认可用的蓝牙设备，例如下面的代码：

```
 BluetoothAdapter cwjBluetoothAdapter =
      BluetoothAdapter.getDefaultAdapter();
 if (cwjBluetoothAdapter == null) {
   // Android 开发网提示大家本机没有找到蓝牙硬件或驱动存在问题
 }
```

经过上述代码后还不知道手机中的蓝牙功能是否被开启，我们可以通过 cwjBluetoothAdapter 的 isEnabled()方法来判断，如果没有开启，可以通过下面的代码提醒用户开启。

```
 if (!cwjBluetoothAdapter.isEnabled()) {
   Intent TurnOnBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
   startActivityForResult(TurnOnBtIntent, REQUEST_ENABLE_BT);
}
```

此时会弹出开启提醒对话框。通过方法 startActivityForResult()发起的 Intent 将会在 onActivityResult()回调方法中获取用户的选择。比如用户单击了 Yes 开启，那么将会收到 RESULT_OK 的结果，如果收到 RESULT_CANCELED 结果，则代表用户不愿意开启蓝牙。

当然也可以通过其他方式来开启。例如用 BluetoothDevice 获取蓝牙服务接口对象，然后用 enable()方法来开启，此方法无须询问用户，但需要用到 android.permission.BLUETOOTH_ADMIN 权限。

该如何判断系统蓝牙的状态呢？建立 BroadcastReceiver 对象，接收 ACTION_STATE_CHANGED 动作，在 EXTRA_STATE 和 EXTRA_PREVIOUS_STATE 包含了现在状态和过去状态，最终的结果定义是 STATE_TURNING_ON 正在开启、STATE_ON 已经开启、STATE_TURNING_OFF 正在关闭和 STATE_OFF 已经关闭。



7.4.2 使用BluetoothSocket类

通过使用 BluetoothSocket 类可以建立和蓝牙有关的通信套接字。蓝牙和 LAN 一样通过 MAC 地址来识别远程设备，建立完通信连接 RFCOMM 通道后以输入、输出流方式进行通信。

1. 连接设备

蓝牙通信和 Java 的 C/S 通信过程一样，也分为 Server 服务器端和 Client 客户端，它们之间使用 BluetoothSocket 类的不同方法来获取数据。

1) 作为服务器端

如果一个设备需要和两个或多个设备连接时，就需要作为一个 Server 来传输。在 Android 中提供了 BluetoothServerSocket 类来处理用户发来的信息，服务器端套接字在 accepted 接收一个客户发来的 BluetoothSocket 时会做出对应的响应。例如下面的代码：

```
private class AcceptThread extends Thread {
  private final BluetoothServerSocket cwjServerSocket;
  public AcceptThread() {
            BluetoothServerSocket tmp = null;
              //使用一个临时对象代替，因为 cwjServerSocket 定义为 final
      try {
            tmp = myAdapter.listenUsingRfcommWithServiceRecord(NAME,
              CWJ_UUID); //服务仅监听
      } catch (IOException e) { }
      cwjServerSocket = tmp;
  }
  public void run() {
      BluetoothSocket socket = null;
            while (true) { //保持连接，直到异常发生或套接字返回
          try {
              socket = cwjServerSocket.accept();  //如果一个连接同意
          } catch (IOException e) {
              break;
          }
          if (socket != null) {
              manageConnectedSocket(socket);
                //管理一个已经连接的 RFCOMM 通道在单独的线程
              cwjServerSocket.close();
              break;
          }
      }
  }
  public void cancel() {  //取消套接字连接，然后线程返回
      try {
          cwjServerSocket.close();
      } catch (IOException e) { }
  }
}
```

在此过程中读者需要注意，服务器在处理任务时需要确保不能阻塞。为了达到不能堵塞的目的，必须使用异步的方法打开一个线程，正如我们上面的演示代码那样。目前 Android 的虚拟机上层没有提供 I/O 模型，所以很有必要在高负载情况下实现性能优化解决方案。

2) 作为客户端

在客户端，为了便于初始化连接到远程设备，首先必须获取本地的 BluetoothDevice 对象，相关的方法在 Android 蓝牙 API 之 BluetoothAdapter 类的两个地方都有讲到，这里不再赘述，相关的演示代码如下：

```
 private class ConnectThread extends Thread {
  private final BluetoothSocket cwjSocket;
  private final BluetoothDevice cwjDevice;
  public ConnectThread(BluetoothDevice device) {
      BluetoothSocket tmp= null;
       cwjDevice= device;
          try {
          tmp= device.createRfcommSocketToServiceRecord(CWJ UUID);
              //客户端创建
      } catch (IOException e) { }
       cwjSocket= tmp;
  }
  public void run() {
           myAdapter.cancelDiscovery();  //取消发现远程设备，这样会降低系统性能
      try {
               cwjSocket.connect();
      } catch (IOException connectException) {
          try {
              cwjSocket.close();
          } catch (IOException closeException) { }
          return;
      }
          manageConnectedSocket(cwjSocket);
            //管理一个已经连接的 RFCOMM 通道在单独的线程
  }
   public void cancel() {
      try {
          cwjSocket.close();
      } catch (IOException e) { }
  }
}
```

2. 管理蓝牙套接字的连接

此过程仍然使用 BluetoothSocket 类来处理具体的数据流。在 Java 中处理数据流很简单，通过 InputSream、OutputSream 和字节数组之间的转化即可处理完毕。由于蓝牙传输中可能存在中断，所以为了防止阻塞，需要设置一个工作者线程。例如下面的演示代码：

```
 private class ConnectedThread extends Thread {
 private final BluetoothSocket cwjSocket;
```



```
    private final InputStream cwjInStream;
    private final OutputStream cwjOutStream;
    public ConnectedThread(BluetoothSocket socket) {
        cwjSocket = socket;
        InputStream tmpIn = null;   //上面定义的为 final，这是使用 temp 临时对象
        OutputStream tmpOut = null;
        try {
            tmpIn = socket.getInputStream();  //使用 getInputStream 作为一个流处理
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }
        cwjInStream = tmpIn;
        cwjOutStream = tmpOut;
    }
    public void run() {
        byte[] buffer = new byte[1024];
        int bytes;
        while (true) {
            try {
                bytes = cwjInStream.read(buffer);
                //传递给 UI 线程
                mHandler.obtainMessage(MESSAGE READ, bytes, -1,
                    buffer).sendToTarget();
               } catch (IOException e) {
                break;
            }
        }
    }
       public void write(byte[] bytes) {
        try {
            cwjOutStream.write(bytes);
        } catch (IOException e) { }
    }
       public void cancel() {
        try {
            cwjSocket.close();
        } catch (IOException e) { }
    }
}
```

在实现具体连接时，在 Android 平台中使用了 Java 标准的输入、输出流来操作实现，BluetoothSocket 提供的 getInputStream()和 getOutputStream()方法可以很好地处理具体的数据。

到此为止，在本书中我们介绍了 Android 中和蓝牙应用相关的两个类的用法。至于其他的类，也都是基于方法实现的，为节省篇幅，不再一一进行详解。

7.4.3　在Android平台开发蓝牙应用的基本步骤

经过前面的学习，了解了在 Android 系统中和蓝牙相关的 API 类。其实从查找蓝牙设备到能够相互通信要经过几个基本步骤，这几个步骤缺一不可。各个步骤的具体说明如下。



1. 设置权限

在文件 AndroidManifest.xml 中声明蓝牙权限。例如下面的代码：

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

2. 启动蓝牙

首先要查看本机是否支持蓝牙，然后获取 BluetoothAdapter 蓝牙适配器对象。例如下面的代码：

```
BluetoothAdapter mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
if(mBluetoothAdapter == null){
       //表明此手机不支持蓝牙
       return;
}
if(!mBluetoothAdapter.isEnabled()){     //蓝牙未开启，则开启蓝牙
            Intent enableIntent = new Intent
                (BluetoothAdapter.ACTION REQUEST ENABLE);
            startActivityForResult(enableIntent, REQUEST ENABLE BT);
}
//…
public void onActivityResult(int requestCode, int resultCode, Intent data){
      if(requestCode == REQUEST ENABLE BT){
            if(requestCode == RESULT OK){
                //蓝牙已经开启
            }
      }
}
```

3. 发现蓝牙设备

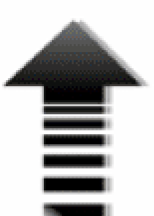
(1) 使本机蓝牙处于可见(即处于易被搜索到的状态)，便于其他设备发现本机蓝牙。演示代码如下：

```
//使本机蓝牙在 300 秒内可被搜索
private void ensureDiscoverable() {
       if (mBluetoothAdapter.getScanMode() !=
          BluetoothAdapter.SCAN MODE CONNECTABLE DISCOVERABLE) {
          Intent discoverableIntent = new Intent
               (BluetoothAdapter.ACTION REQUEST DISCOVERABLE);
          discoverableIntent.putExtra(BluetoothAdapter.
               EXTRA DISCOVERABLE DURATION, 300);
          startActivity(discoverableIntent);
       }
}
```

(2) 查找已经配对的蓝牙设备，即以前已经配对过的设备。演示代码如下：

```
       Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
        if (pairedDevices.size() > 0) {
```





```
            findViewById(R.id.title paired devices).setVisibility(View.VISIBLE);
            for (BluetoothDevice device : pairedDevices) {
                //device.getName() +" "+ device.getAddress());
            }
        } else {
            mPairedDevicesArrayAdapter.add("没有找到已匹配的设备");
        }
```

(3) 通过 mBluetoothAdapter.startDiscovery() 来搜索设备，在此需要注册一个 BroadcastReceiver 来获得这个搜索结果。即先注册再获取信息，然后进行处理。演示代码如下：

```
//注册，当一个设备被发现时调用 onReceive
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION FOUND);
        this.registerReceiver(mReceiver, filter);
//当搜索结束后调用 onReceive
filter = new IntentFilter(BluetoothAdapter.ACTION DISCOVERY FINISHED);
        this.registerReceiver(mReceiver, filter);
//…
private BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if(BluetoothDevice.ACTION FOUND.equals(action)){
                 BluetoothDevice device = intent.getParcelableExtra
                     (BluetoothDevice.EXTRA DEVICE);
                // 已经配对的则跳过
               if (device.getBondState() != BluetoothDevice.BOND_BONDED)
{
                     mNewDevicesArrayAdapter.add(device.getName() + "\n" +
                          device.getAddress()); //保存设备地址与名字
               }
            }else if (BluetoothAdapter.ACTION DISCOVERY FINISHED.
                 equals(action)) { //搜索结束
              if (mNewDevicesArrayAdapter.getCount() == 0) {
                 mNewDevicesArrayAdapter.add("没有搜索到设备");
              }
            }
        }
};
```

4. 建立连接

当查找到蓝牙设备后，接下来需要建立本机与其他设备之间的连接。一般在使用本机搜索其他蓝牙设备时，本机可以作为一个服务器端来接收其他设备的连接。启动一个服务器端的线程，循环等待客户端的连接，这与 ServerSocket 极为相似，此线程在准备连接之前启动。演示代码如下：

```
//UUID 可以看作一个端口号
private static final UUID MY UUID =
UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
```

```
    //像一个服务器一样时刻监听是否有连接建立
    private class AcceptThread extends Thread{
    private BluetoothServerSocket serverSocket;

    public AcceptThread(boolean secure){
        BluetoothServerSocket temp = null;
        try {
            temp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(
                        NAME_INSECURE, MY_UUID);
        } catch (IOException e) {
                Log.e("app", "listen() failed", e);
            }
        serverSocket = temp;
    }

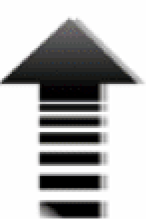
    public void run(){
        BluetoothSocket socket=null;
        while(true){
            try {
                    socket = serverSocket.accept();
                } catch (IOException e) {
                    Log.e("app", "accept() failed", e);
                  break;
                }
        }
        if(socket!=null){
            //此时可以新建一个数据交换线程，把此 Socket 传进去
        }
    }

    //取消监听
    public void cancel(){
           try {
               serverSocket.close();
           } catch (IOException e) {
               Log.e("app", "Socket Type" + socketType + "close() of
server failed", e);
           }
    }
}
```

5. 交换数据

当搜索到蓝牙设备后，接下来可以获取设备的地址，通过此地址获取一个 BluetoothDeviced 对象，可以将其看作是一个客户端，通过对象 device.createRfcommSocketToServiceRecord (MY_UUID)和一个 UUID 可与服务器建立连接，获取另一个 Socket 对象。因为此服务器端与客户端各有一个 Socket 对象，所以此时它们可以互相交换数据了。演示代码如下：

```
    //另一个设备去连接本机，相当于客户端
    private class ConnectThread extends Thread{
    private BluetoothSocket socket;
```



```
    private BluetoothDevice device;
    public ConnectThread(BluetoothDevice device,boolean secure){
        this.device = device;
        BluetoothSocket tmp = null;
        try {
                tmp = device.createRfcommSocketToServiceRecord(MY UUID SECURE);
            } catch (IOException e) {
                 Log.e("app", "create() failed", e);
            }
    }

    public void run(){
        mBluetoothAdapter.cancelDiscovery();    //取消设备查找
        try {
                socket.connect();
            } catch (IOException e) {
                try {
                    socket.close();
                } catch (IOException e1) {
                     Log.e("app", "unable to close() "+
                                  " socket during connection failure", e1);
                }
                connetionFailed();  //连接失败
                return;
            }
          //此时可以新建一个数据交换线程，把此 Socket 传进去
    }

      public void cancel() {
            try {
                socket.close();
            } catch (IOException e) {
                Log.e("app", "close() of connect  socket failed", e);
            }
        }
    }
```

6. 建立数据通信线程

此阶段的任务是读取数据。演示代码如下：

```
//建立连接后，进行数据通信的线程
   private class ConnectedThread extends Thread{
    private BluetoothSocket socket;
    private InputStream inStream;
    private OutputStream outStream;

    public ConnectedThread(BluetoothSocket socket){

        this.socket = socket;
        try {
```




```
        //获得输入输出流
            inStream = socket.getInputStream();
            outStream = socket.getOutputStream();
        } catch (IOException e) {
            Log.e("app", "temp sockets not created", e);
        }
}

public void run(){
    byte[] buff = new byte[1024];
    int len=0;
    //读数据需不断监听，写不需要
    while(true){
        try {
                len = inStream.read(buff);
                //把读取到的数据发送给 UI 进行显示
                Message msg = handler.obtainMessage
                    (BluetoothChat.MESSAGE_READ, len, -1, buff);
                msg.sendToTarget();
            } catch (IOException e) {
                Log.e("app", "disconnected", e);
             connectionLost();  //失去连接
             start();           //重新启动服务器
             break;
            }
    }
}


public void write(byte[] buffer) {
        try {
            outStream.write(buffer);
            handler.obtainMessage(BluetoothChat.MESSAGE_WRITE, -1, -1, buffer)
                    .sendToTarget();
        } catch (IOException e) {
            Log.e("app", "Exception during write", e);
        }
    }
    public void cancel() {
        try {
            socket.close();
        } catch (IOException e) {
            Log.e("app", "close() of connect socket failed", e);
        }
    }
}
```

到此为止，一个基本的蓝牙通信操作已经全部完成。读者在开发此类项目时，只需按照上述流程进行即可。



7.5　开发一个遥控器——蓝牙控制玩具车

开发蓝牙项目是一件很麻烦的事情，这里说的麻烦不仅仅是类多、函数多，而且在测试时不能用模拟器来实现，我们需要真机，并且是两部具有蓝牙模块的设备。笔者为了节省成本，在淘宝网上网购了一个蓝牙模块，开始了我们的这个实例之旅。

实　例	功　能	源码路径
实例 7-1	通过蓝牙遥控指挥玩具车	下载路径:\daima\7\lanya

(1) 将网购的蓝牙模块放在一辆玩具车上，并为其接通电源。

(2) 打开 Eclipse 新建一个 Android 工程文件。

(3) 编写布局文件 main.xml。在其中插入 5 个控制按钮，分别实现对玩具车的向前、左转、右转、后退和停止的控制。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
android:id="@+id/widget0"
android:layout width="fill parent"
android:layout height="fill parent"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<Button
android:id="@+id/btnF"
android:layout width="100px"
android:layout height="60px"
android:text="向前"
android:layout x="130px"
android:layout_y="62px"
>
</Button>
<Button
android:id="@+id/btnL"
android:layout width="100px"
android:layout height="60px"
android:text="左转"
android:layout x="20px"
android:layout y="152px"
>
</Button>
<Button
android:id="@+id/btnR"
android:layout width="100px"
android:layout height="60px"
android:text="右转"
android:layout x="240px"
android:layout_y="152px"
```

```
>
</Button>
<Button
android:id="@+id/btnB"
android:layout_width="100px"
android:layout_height="60px"
android:text="后退"
android:layout_x="130px"
android:layout_y="242px"
>
</Button>
<Button
android:id="@+id/btnS"
android:layout_width="100px"
android:layout_height="60px"
android:text="停止"
android:layout_x="130px"
android:layout_y="152px"
>
</Button>
</AbsoluteLayout>
```

(4) 编写蓝牙程序控制文件 lanya.java。其实现原理和 7.4 节中讲解的步骤一致。具体实现流程如下。

① 定义类 lanya，然后设置 5 个按钮对象。具体代码如下：

```
public class lanya extends Activity {
    private static final String TAG = "THINBTCLIENT";
   private static final boolean D = true;
   private BluetoothAdapter mBluetoothAdapter = null;
   private BluetoothSocket btSocket = null;

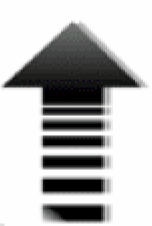
   private OutputStream outStream = null;
   Button mButtonF;
   Button mButtonB;
   Button mButtonL;
   Button mButtonR;
   Button mButtonS;
```

② 赋值蓝牙设备上的标准串行和要连接的蓝牙设备 MAC 地址，具体代码如下：

```
   private static final UUID MY_UUID = UUID.fromString("00011101-0000-
     1000-807-00805F9B34FB");//蓝牙设备上的标准串行
   private static String address = "00:11:03:21:00:42"; // <==要连接的蓝牙
     设备 MAC 地址
```

③ 编写单击“向前”按钮的处理事件。具体代码如下：

```
   @Override
   public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);
```



```
//向前
mButtonF=(Button)findViewById(R.id.btnF);
mButtonF.setOnTouchListener(new Button.OnTouchListener(){

      @Override
      public boolean onTouch(View v, MotionEvent event) {
          // TODO Auto-generated method stub
          String message;
          byte[] msgBuffer;
          int action = event.getAction();
          switch(action)
          {
          case MotionEvent.ACTION_DOWN:
          try {
          outStream = btSocket.getOutputStream();
         } catch (IOException e) {
            Log.e(TAG, "ON RESUME: Output stream creation
                  failed.", e);
         }
         message = "1";
        msgBuffer = message.getBytes();
         try {
          outStream.write(msgBuffer);
         } catch (IOException e) {
                Log.e(TAG, "ON RESUME: Exception during write.", e);
         }
          break;
          case MotionEvent.ACTION_UP:
              try {
              outStream = btSocket.getOutputStream();
             } catch (IOException e) {
                Log.e(TAG, "ON RESUME: Output stream creation
                      failed.", e);
             }
             message = "0";
             msgBuffer = message.getBytes();
             try {
              outStream.write(msgBuffer);
             } catch (IOException e) {
                    Log.e(TAG, "ON RESUME: Exception during
                          write.", e);
             }
              break;
          }
          return false;
      }
});
```

④ 编写单击“后退”按钮的处理事件，具体代码如下：

```
mButtonB=(Button)findViewById(R.id.btnB);
mButtonB.setOnTouchListener(new Button.OnTouchListener(){
```

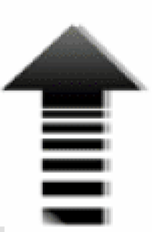
```
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            // TODO Auto-generated method stub
            String message;
            byte[] msgBuffer;
            int action = event.getAction();
            switch(action)
            {
            case MotionEvent.ACTION_DOWN:
            try {
            outStream = btSocket.getOutputStream();
          } catch (IOException e) {
              Log.e(TAG, "ON RESUME: Output stream creation
                    failed.", e);
          }
          message = "3";
         msgBuffer = message.getBytes();
          try {
            outStream.write(msgBuffer);
          } catch (IOException e) {
                  Log.e(TAG, "ON RESUME: Exception during write.", e);
          }
            break;

            case MotionEvent.ACTION_UP:
                try {
                outStream = btSocket.getOutputStream();
              } catch (IOException e) {
                  Log.e(TAG, "ON RESUME: Output stream creation
                        failed.", e);
              }
              message = "0";
              msgBuffer = message.getBytes();
              try {
                outStream.write(msgBuffer);
              } catch (IOException e) {
                      Log.e(TAG, "ON RESUME: Exception during
                             write.", e);
              }
                break;
            }

            return false;
        }
    });
```

⑤ 编写单击“左转”按钮的处理事件。具体代码如下：

```
    mButtonL=(Button)findViewById(R.id.btnL);
    mButtonL.setOnTouchListener(new Button.OnTouchListener(){
        @Override
```



```
            public boolean onTouch(View v, MotionEvent event) {
                // TODO Auto-generated method stub
                String message;
                byte[] msgBuffer;
                int action = event.getAction();
                switch(action)
                {
                case MotionEvent.ACTION_DOWN:
                try {
                outStream = btSocket.getOutputStream();
               } catch (IOException e) {
                  Log.e(TAG, "ON RESUME: Output stream creation
                         failed.", e);
               }
               message = "2";
              msgBuffer = message.getBytes();
               try {
                outStream.write(msgBuffer);
               } catch (IOException e) {
                      Log.e(TAG, "ON RESUME: Exception during write.", e);
               }
                break;

                case MotionEvent.ACTION_UP:
                    try {
                    outStream = btSocket.getOutputStream();
                   } catch (IOException e) {
                      Log.e(TAG, "ON RESUME: Output stream creation
                             failed.", e);
                   }
                   message = "0";
                   msgBuffer = message.getBytes();
                   try {
                    outStream.write(msgBuffer);
                   } catch (IOException e) {
                          Log.e(TAG, "ON RESUME: Exception during
                                write.", e);
                   }
                    break;
                }

                return false;

            }
        });
```

⑥ 编写单击“右转”按钮的处理事件。具体代码如下：

```
        mButtonR=(Button)findViewById(R.id.btnR);
        mButtonR.setOnTouchListener(new Button.OnTouchListener(){
              @Override
```

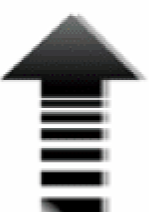
```
            public boolean onTouch(View v, MotionEvent event) {
                // TODO Auto-generated method stub
                String message;
                byte[] msgBuffer;
                int action = event.getAction();
                switch(action)
                {
                case MotionEvent.ACTION DOWN:
                try {
                outStream = btSocket.getOutputStream();
               } catch (IOException e) {
                  Log.e(TAG, "ON RESUME: Output stream creation
                         failed.", e);
               }
               message = "4";
              msgBuffer = message.getBytes();
               try {
                outStream.write(msgBuffer);
               } catch (IOException e) {
                      Log.e(TAG, "ON RESUME: Exception during write.", e);
               }
                break;

                case MotionEvent.ACTION UP:
                    try {
                    outStream = btSocket.getOutputStream();
                   } catch (IOException e) {
                      Log.e(TAG, "ON RESUME: Output stream creation
                             failed.", e);
                   }
                   message = "0";
                   msgBuffer = message.getBytes();
                   try {
                    outStream.write(msgBuffer);
                   } catch (IOException e) {
                          Log.e(TAG, "ON RESUME: Exception during
                                write.", e);
                   }
                    break;
                }

                return false;
            }
        });
```

⑦ 编写单击“停止”按钮的处理事件。具体代码如下：

```
        mButtonS=(Button)findViewById(R.id.btnS);
        mButtonS.setOnTouchListener(new Button.OnTouchListener(){
            @Override
            public boolean onTouch(View v, MotionEvent event) {
```



```
                    // TODO Auto-generated method stub
                    if(event.getAction()==MotionEvent.ACTION DOWN)
                     try {
                        outStream = btSocket.getOutputStream();
                      } catch (IOException e) {
                         Log.e(TAG, "ON RESUME: Output stream creation
                               failed.", e);
                      }
                      String message = "0";
                      byte[] msgBuffer = message.getBytes();
                      try {
                       outStream.write(msgBuffer);
                      } catch (IOException e) {
                            Log.e(TAG, "ON RESUME: Exception during
                                  write.", e);
                      }
                    return false;
                }
          });
          if (D)
            Log.e(TAG, "+++ ON CREATE +++");
              mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
          if (mBluetoothAdapter == null) {
            Toast.makeText(this, "蓝牙设备不可用，请打开蓝牙！",
                  Toast.LENGTH LONG).show();
                  finish();
                  return;
          }
          if (!mBluetoothAdapter.isEnabled()) {
                  Toast.makeText(this,  "请打开蓝牙并重新运行程序！",
                     Toast.LENGTH LONG).show();
                  finish();
                  return;
          }
          if (D)
          Log.e(TAG, "+++ DONE IN ON CREATE, GOT LOCAL BT ADAPTER +++");
    }
```

⑧ 通过套接字建立蓝牙连接，如果失败则输出失败提示。主要代码如下：

```
    @Override
    public void onStart() {
          super.onStart();
          if (D) Log.e(TAG, "++ ON START ++");
    }
    @Override
    public void onResume() {
          super.onResume();
          if (D) {
            Log.e(TAG, "+ ON RESUME +");
           Log.e(TAG, "+ ABOUT TO ATTEMPT CLIENT CONNECT +");
```




```
        }
        DisplayToast("正在尝试连接智能小车，请稍后・・・・");
        BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
        try {
           btSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
             DisplayToast("套接字创建失败！");
        }
        DisplayToast("成功连接智能小车！可以开始操控了~~~");
        mBluetoothAdapter.cancelDiscovery();
        try {
                btSocket.connect();
                DisplayToast("连接成功建立，数据连接打开！");

        } catch (IOException e) {
                try {
                   btSocket.close();
                } catch (IOException e2) {

                        DisplayToast("连接没有建立，无法关闭套接字！");
                }
        }
        if (D)
          Log.e(TAG, "+ ABOUT TO SAY SOMETHING TO SERVER +");
}
@Override
public void onPause() {
        super.onPause();
        if (D)
          Log.e(TAG, "- ON PAUSE -");
        if (outStream != null) {
                try {
                        outStream.flush();
                } catch (IOException e) {
                        Log.e(TAG, "ON PAUSE: Couldn't flush output
                            stream.", e);
                }
        }
        try {
                btSocket.close();
        } catch (IOException e2) {

                DisplayToast("套接字关闭失败！");
        }
}
@Override
public void onStop() {
        super.onStop();
        if (D)Log.e(TAG, "-- ON STOP --");
}
@Override
```

```
    public void onDestroy() {
            super.onDestroy();
            if (D) Log.e(TAG, "--- ON DESTROY ---");
    }
    public void DisplayToast(String str)
    {
     Toast toast=Toast.makeText(this, str, Toast.LENGTH LONG);
     toast.setGravity(Gravity.TOP, 0, 220);
     toast.show();
     }
}
```

(5) 在文件 AndroidManifest.xml 中声明蓝牙权限，对应代码如下：

```
<uses-permission android:name="android.permission.BLUETOOTH ADMIN" />
 <uses-permission android:name="android.permission.BLUETOOTH" />
   <application android:icon="@drawable/icon" android:label=
        "@string/app name">
      <activity android:name=".lanya"
              android:label="@string/app name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
```

到此为止，我们的蓝牙控制玩具车的实例就介绍完毕了。本实例的实现比较简单，难度大的是双向控制，即实现每个设备都可以操控另外一个设备的功能，此时需要有蓝牙功能的电脑或 Android 手机来完成测试了。在模拟器中因为不具备蓝牙设备，程序执行后会显示“蓝牙设备不可用，请打开蓝牙！”的提示，如图 7-3 所示。

图 7-3　模拟器的运行效果

Android

第8章 在Android中开发Wi-Fi应用

Wi-Fi 是一种可以将个人电脑、手持设备(如 PDA、手机)等终端以无线方式互相连接的技术。Wi-Fi 是一个无线网路通信技术的品牌，由 Wi-Fi 联盟(Wi-Fi Alliance)所持有。目的是改善基于 IEEE 802.11 标准的无线网路产品之间的互通性。现在一般人会把 Wi-Fi 及 IEEE 802.11 混为一谈。甚至直接把 Wi-Fi 等同于无线网际网路。本章将简要介绍在 Android 平台中开发 Wi-Fi 相关应用的基本知识。

8.1 了解 Wi-Fi 系统的结构

Wi-Fi 系统比较复杂，但是却比较常用，要想完全掌握 Wi-Fi 应用开发技术，我们需要从底层做起，需要先了解它的底层结构。本节将简要讲解 Wi-Fi 系统底层结构的基本知识。

8.1.1 Wi-Fi概述

在 Android 系统中，存在一个无线控制模块。打开方式如下：依次选择 Menu | Settings | Wireless$networks | Wi-Fi settings 命令，打开如图 8-1 所示的界面。

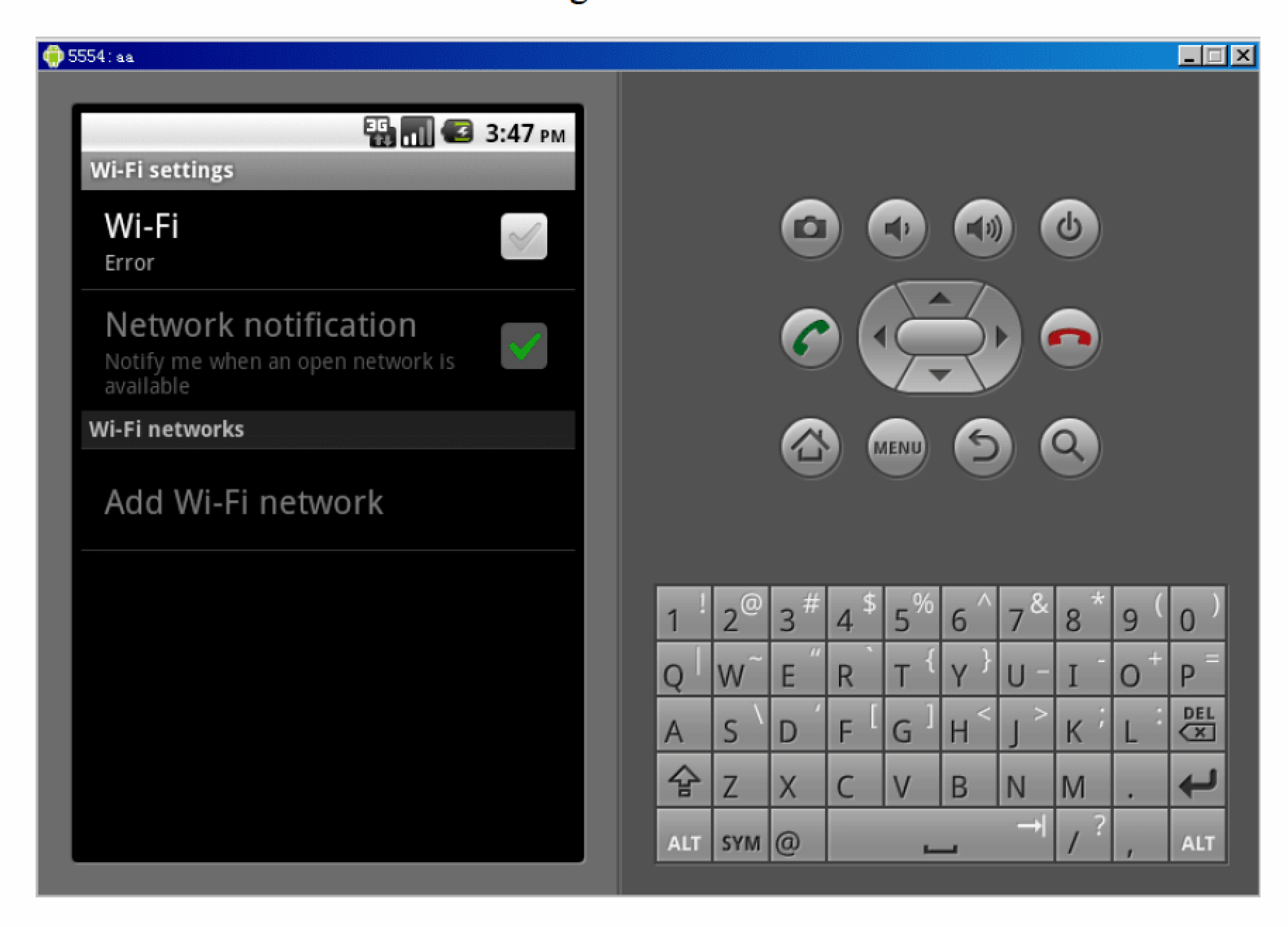


图 8-1 Wi-Fi控制界面

8.1.2 Wi-Fi层次结构

Wi-Fi 系统的上层接口包括数据部分和控制部分。数据部分通常是一个和以太网卡类似的网络设备，控制部分用于实现接入点操作和安全验证处理。

在软件层，Wi-Fi 系统包括 Linux 内核程序和协议，还包括本地部分、Java 框架类。Wi-Fi 系统向 Java 应用程序层提供了控制类的接口。

Android 平台中 Wi-Fi 系统的基本层次结构如图 8-2 所示。

由图 8-2 可知，Android 平台中 Wi-Fi 系统从上到下主要包括 Java 框架类、Android 适配器库、wpa_supplicant 守护进程、驱动程序和协议，这几部分的系统结构如图 8-3 所示。

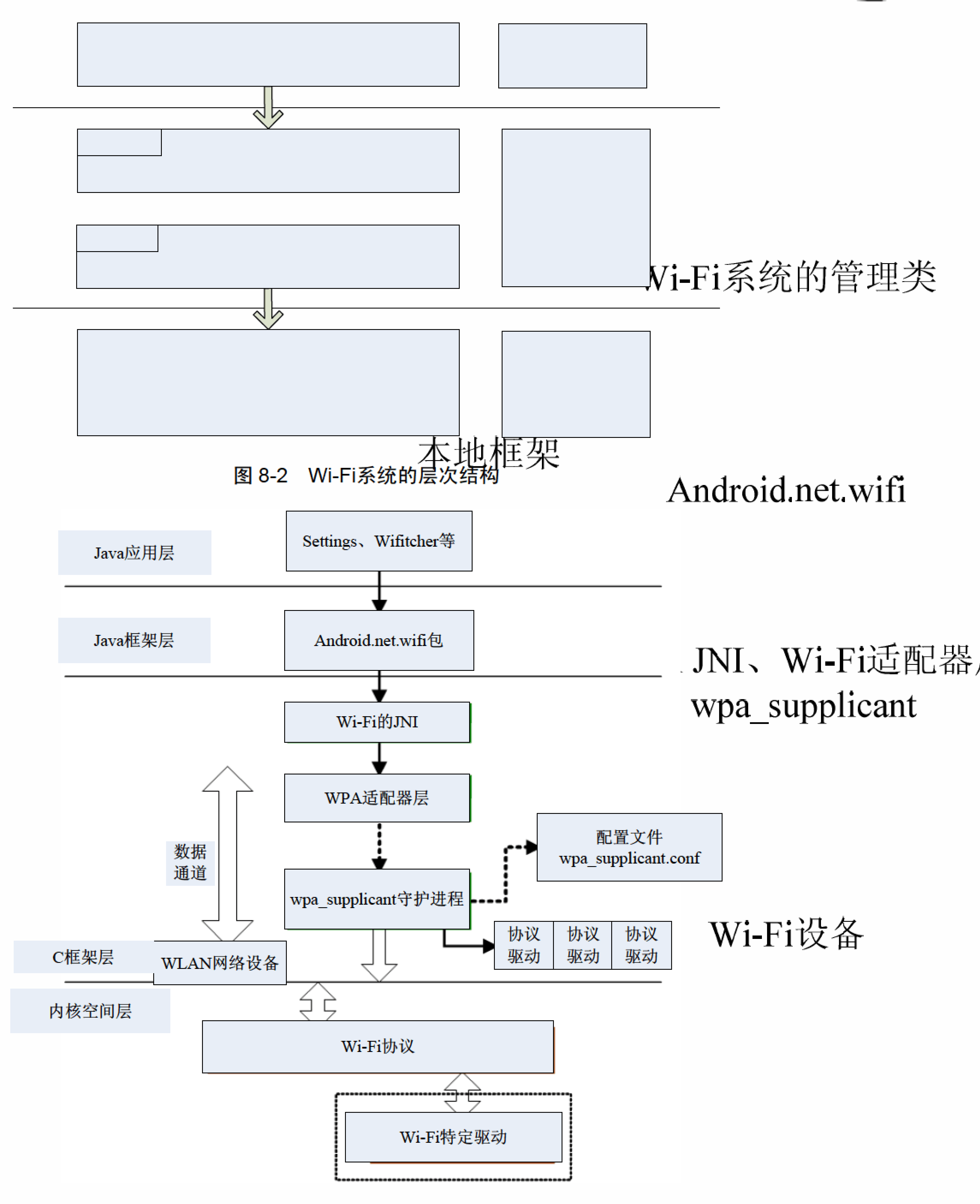


图 8-2　Wi-Fi系统的层次结构

图 8-3　Wi-Fi的系统结构

图 8-3 中各个部分的具体说明如下。

(1) Wi-Fi 用户空间的程序和库，对应路径如下：

```
external/wpa_supplicant/
```

在此生成库 libwpaclient.so 和守护进程 wpa_supplicant。

(2) Wi-Fi 管理库，即适配器库，通过调用库 libwpaclient.so 成为 wpa_supplicant 在 Android 中的客户端。对应路径如下：

```
hardware/libhardware_legary/wifi/
```

(3) JNI 部分的对应路径如下：

```
frameworks/base/core/jni/android_net_wifi_Wifi.cpp
```

(4) Java 框架部分的对应路径如下：

```
frameworks/base/services/java/com/android/server/
frameworks/base/wifi/java/android/net/wifi/
```

在 android.net.wifi 将作为 Android 平台的 API 供 Java 应用程序层使用。

(5) Wi-Fi Settings 应用程序的对应路径如下：

```
packages/apps/Settings/src/com/android/settings/wifi/
```

8.1.3　Wi-Fi在Android和Linux中的差异

我们先看 Wi-Fi 在 Android 中是如何工作的：Android 使用一个修改版 wpa_supplicant 作为 daemon 来控制 Wi-Fi，代码位于如下目录中。

```
external/wpa_supplicant
```

wpa_supplicant 是通过 socket 与文件 hardware/libhardware_legacy/wifi/wifi.c 进行通信。UI 通过 android.net.wifi package(frameworks/base/wifi/java/android/net/wifi/)发送命令给文件 wifi.c。相应的 JNI 实现位于文件 frameworks/base/core/jni/android_net_wifi_Wifi.cpp 中，更高一级的网络管理位于如下目录中。

```
frameworks/base/core/java/android/net
```

在 Android 中的无线局域网部分是标准的系统，并且针对特定的硬件平台，所以需要移植和改动的内容并不多。在 Linux 内核中有 Wi-Fi 的标准协议，不同硬件平台的差异仅仅体现在 Wi-Fi 芯片驱动程序。除了这些芯片级驱动的差异外，在 Android 中实现其他无线局域网部分的方法在 Linux 内核中已经给出了具体方法。

而在 Android 用户空间中，使用了标准的 wpa_supplicant 守护进程，这也是一个标准的实现，所以无须我们为 Wi-Fi 增加单独的硬件抽象层代码，只需进行简单的配置工作即可。

8.2　分析 Wi-Fi 源码

要想掌握 Wi-Fi 的开发原理，我们需要分析 Android 中的 Wi-Fi 源码并了解其核心构造，这样才能对 Wi-Fi 应用开发做到游刃有余。本节将简要介绍开源 Android 中和 Wi-Fi



相关的代码。

8.2.1　本地部分

本地实现部分主要包括 wpa_supplicant 以及 wpa_supplicant 适配层。WPA 是 Wi-Fi Protected Access 的缩写，中文含义为“WiFi 网络安全存取”。WPA 是一种基于标准的可互操作的 WLAN 安全性增强解决方案，可大大增强现有以及未来无线局域网系统的数据保护和访问控制水平。

wpa_supplicant 适配层是通用的 wpa_supplicant 的封装，在 Android 中作为 Wi-Fi 部分的硬件抽象层来使用。wpa_supplicant 适配层主要用于封装与 wpa_supplicant 守护进程的通信，以提供给 Android 框架使用。它实现了加载、控制和消息监控等功能。wpa_supplicant 适配层的头文件如下：

```
hardware/libhardware_legacy/include/hardware_legacy/wifi.h
```

wpa_supplicant 的标准结构框图如图 8-4 所示。

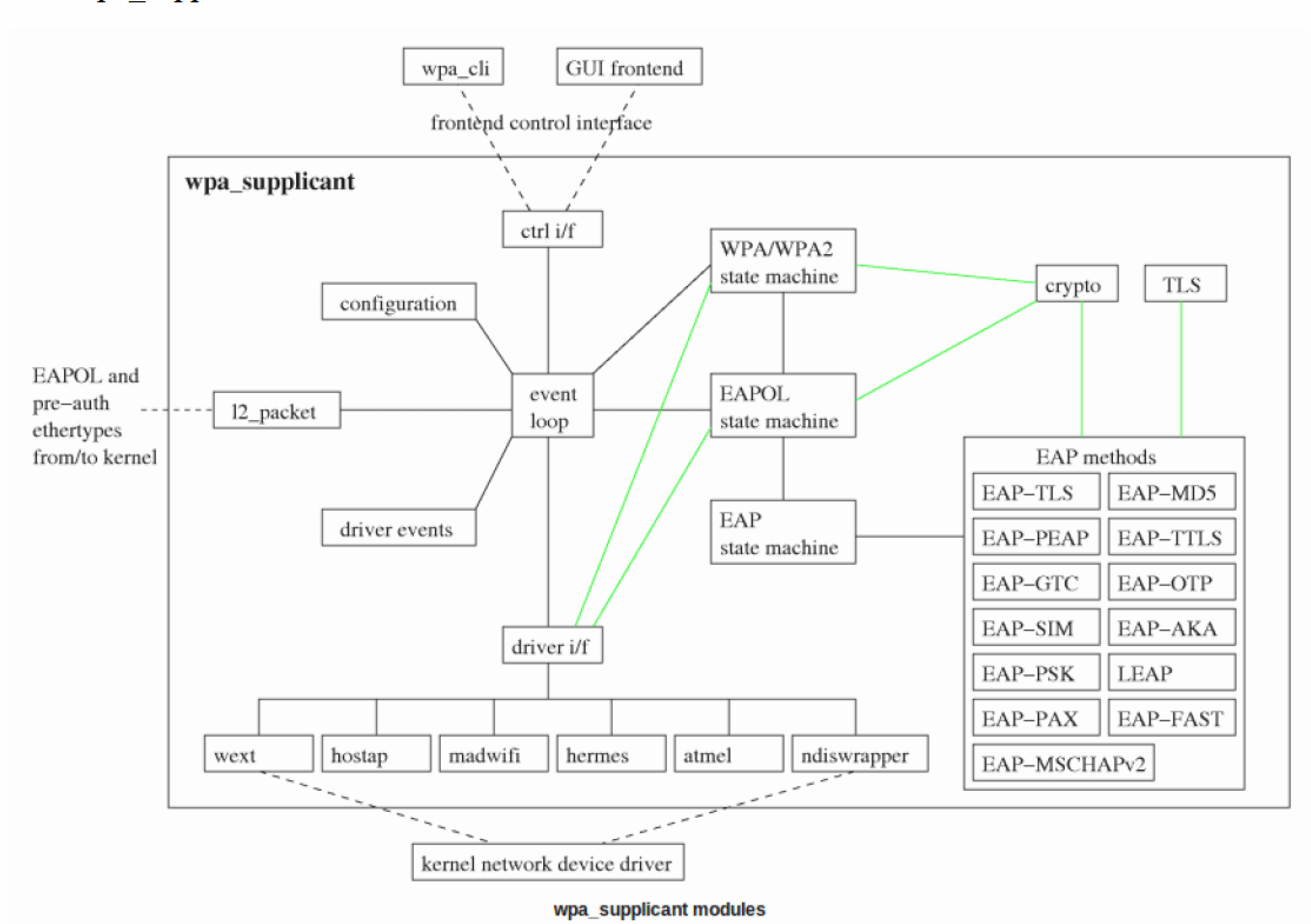


图 8-4　wpa_supplicant的标准结构框图

我们重点关注框图的下半部分，即 wpa_supplicant 是如何与 Driver 进行联系的。假设整个过程以 AP 发出 SCAN 命令为主线。由于现在大部分 Wi-Fi Driver 都支持 wext，所以就假设我们的设备走的是 wext 这条线，其实用 ndis 也一样，整个流程也差不多。

首先要说的是，在文件 Driver.h 文件中存在一个名为 wpa_driver_ops 的结构体，这个

结构体在 Driver.c 中被声明如下：

```
#ifdef CONFIG DRIVER WEXT
extern struct wpa_driver_ops wpa_driver_wext_ops;
```

然后在文件 driver_wext.c 中添加了该结构体的成员，代码如下：

```
const struct wpa driver ops wpa driver wext ops = {
    .name = "wext",
    .desc = "Linux wireless extensions (generic)",
    .get bssid = wpa driver wext get bssid,
    .get ssid = wpa driver wext get ssid,
    .set wpa = wpa driver wext set wpa,
    .set key = wpa driver wext set key,
    .set countermeasures = wpa driver wext set countermeasures,
    .set drop unencrypted = wpa driver wext set drop unencrypted,
    .scan = wpa driver wext scan,
    .combo scan = wpa driver wext combo scan,
    .get scan results2 = wpa driver wext get scan results,
    .deauthenticate = wpa driver wext deauthenticate,
    .disassociate = wpa driver wext disassociate,
    .set_mode = wpa_driver_wext_set_mode,
    .associate = wpa driver wext associate,
    .set auth alg = wpa driver wext set auth alg,
    .init = wpa driver wext init,
    .deinit = wpa driver wext deinit,
    .add pmkid = wpa driver wext add pmkid,
    .remove pmkid = wpa driver wext remove pmkid,
    .flush pmkid = wpa driver wext flush pmkid,
    .get capa = wpa driver wext get capa,
    .set operstate = wpa driver wext set operstate,
#ifdef ANDROID
    .driver cmd = wpa driver priv driver cmd,
#endif
};
```

上述成员其实都是驱动和 wpa_supplicant 的接口。以 SCAN 为例的代码如下：

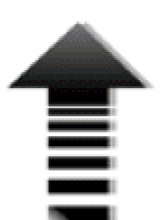
```
int wpa_driver_wext_scan(void *priv, const u8 *ssid, size_t ssid_len)
```

通过如下代码可以看出 wpa_supplicant 是通过 IOCTL 来调用 SOCKET 与 DRIVER 进行通信的，并给 DRIVER 下达 SIOCSIWSCAN 命令。

```
if (ioctl(drv->ioctl_sock, SIOCSIWSCAN, &iwr) < 0)
```

这样，当一个命令从 AP 到 Framework，然后到 C++本地库，再到 wpa_supplicant 适配层，最后 wpa_supplicant 下 CMD 给 Driver 的路线就打通了。

因为 Wi-Fi 模块是采用 SDIO 总线来控制的，所以应该先记录下 Client Driver 的 SDIO 部分的结构。此部分的 SDIO 分为三层，分别是 SdioDrv、SdioAdapter 和 SdioBusDrv。其中 SdioBusDrv 是 Client Driver 中 SDIO 与 Wi-Fi 模块的接口，SdioAdapter 是 SdioDrv 和 SdioBusDrv 之间的适配层，SdioDrv 是 Client Driver 中 SDIO 与 Linux Kernel 中的 MMC



SDIO 的接口。这三部分只需要关注一下 SdioDrv 就可以了，另外两层都只是对它的封装。

在 SdioDrv 中提供了下面的功能。

```
static struct sdio driver tiwlan sdio drv = {
    .probe = tiwlan sdio probe,
    .remove = tiwlan sdio remove,
    .name = "sdio tiwlan",
    .id table = tiwl12xx devices,
};
int sdioDrv EnableFunction(unsigned int uFunc)
int sdioDrv_EnableInterrupt(unsigned int uFunc)
```

SDIO 的读写实际上调用了 MMC\Core 中的如下功能函数：

```
static int mmc_io_rw_direct_host()
```

SDIO 功能部分读者只需简单了解即可，一般 host 部分芯片厂商都会提供完整的解决方案。我们的主要任务还是 Wi-Fi 模块。

首先来看 Wi-Fi 模块的入口函数 wlanDrvIf_ModuleInit()，在此调用了 wlanDrvIf_Create()。主要代码如下：

```
static int wlanDrvIf Create (void)
{
    TWlanDrvIfObj *drv; //这个结构体为代表设备，包含 Linux 网络设备结构体 net device
    pDrvStaticHandle = drv;
    drv->pWorkQueue = create singlethread workqueue (TIWLAN DRV NAME);
        //创建了工作队列
    rc = wlanDrvIf SetupNetif (drv);
    drv->wl sock = netlink kernel create( NETLINK USERSOCK, 0, NULL, NULL,
        THIS MODULE );
    // 创建了接受 wpa_supplicant 的 Socket 接口
    rc = drvMain Create (drv,
                    &drv->tCommon.hDrvMain,
                    &drv->tCommon.hCmdHndlr,
                    &drv->tCommon.hContext,
                    &drv->tCommon.hTxDataQ,
                    &drv->tCommon.hTxMgmtQ,
                    &drv->tCommon.hTxCtrl,
                    &drv->tCommon.hTWD,
                    &drv->tCommon.hEvHandler,
                    &drv->tCommon.hCmdDispatch,
                    &drv->tCommon.hReport,
                    &drv->tCommon.hPwrState);
    rc = hPlatform initInterrupt (drv, (void*)wlanDrvIf HandleInterrupt);
   return 0;
}
```

在调用完 wlanDrvIf_Create()函数后，实际上 Wi-Fi 模块的初始化就结束了。

下面分析是如何初始化的。先看 wlanDrvIf_SetupNetif (drv)函数的主体，对应代码如下：

```
static int wlanDrvIf SetupNetif (TWlanDrvIfObj *drv)
{
   struct net device *dev;
   int res;
   dev = alloc_etherdev (0); //申请 Linux 网络设备
   if (dev == NULL)
     ether setup (dev);      //建立网络接口，这两个都是 Linux 网络设备驱动的标准函数
   dev->netdev ops = &wlan netdev ops;
   wlanDrvWext Init (dev);
   res = register netdev (dev);
   hPlatform SetupPm(wlanDrvIf Suspend, wlanDrvIf Resume, pDrvStaticHandle);
}
```

在此初始化了 wlanDrvWext_Init(dev)，说明 wpa_supplicant 与 Driver 直接的联系是走的 wext 这条路。也就是说 event 的接收，处理也应该是在 wext 部分来实现的。确定此论点之后，剩下的工作量立刻减少了三分之一。接下来需要注册网络设备 dev，在 wlan_netdev_ops 中的定义代码如下：

```
static const struct net device ops wlan netdev ops = {
       .ndo open = wlanDrvIf Open,
       .ndo stop = wlanDrvIf Release,
       .ndo do ioctl = NULL,
       .ndo_start_xmit = wlanDrvIf_Xmit,
       .ndo get stats = wlanDrvIf NetGetStat,
       .ndo validate addr = NULL,
};
```

上述代码名字对应的都是 Linux 网络设备驱动的命令字，最后需要调用“rc=drvMain_CreateI”，通过此函数完成相关模块的初始化工作。

8.2.2　JNI部分

Android 中的 Wi-Fi 系统的 JNI 部分实现的源码文件如下：

```
frameworks/base/core/jni/android_net_wifi_Wifi.cpp
```

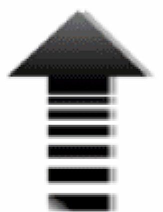
JNI 层的接口注册到 Java 层的源代码文件如下：

```
frameworks/base/wifi/java/android/net/wifi/WifiNative.java
```

WifiNative 将为 WifiService、WifiStateTracker、WifiMonitor 等几个 Wi-Fi 框架内部组件提供底层操作支持。

此处实现的本地函数都是通过调用 wpa_supplicant 适配层的接口来实现的(包含适配层的头文件 wifi.h)。wpa_supplicant 适配层是通用的 wpa_supplicant 的封装，在 Android 中作为 Wi-Fi 部分的硬件抽象层来使用。wpa_supplicant 适配层主要用于封装与 wpa_supplicant 守护进程的通信，以提供给 Android 框架使用。它实现了加载、控制和消息监控等功能。wpa_supplicant 适配层的头文件如下：

```
hardware/libhardware_legacy/include/hardware_legacy/wifi.h
```



文件 wifi.h 是 Wi-Fi 适配器层对 JNI 部分的接口，其中包含了一些加载和连接的控制接口，主要包括如下两个接口。

- wifi_command()：负责将命令发送到 Wi-Fi 下层。
- wifi_wait_for_event()：负责事件进入通道，此函数将被阻塞，直到收到一个 Wi-Fi 事件为止，并且以字符串的形式返回。

在文件 wifi.h 中定义上述接口的代码如下：

```
int wifi command(const char *command, char *reply, size t *reply len);
int wifi_wait_for_event(char *buf, size_t len);
```

在文件 wifi.c 中实现了上述两个接口，具体代码如下：

```
int wifi command(const char *command, char *reply, size t *reply len)
{
   return wifi send command(ctrl conn, command, reply, reply len);
}
int wifi wait for event(char *buf, size t buflen)
{
   size t nread = buflen - 1;
   int fd;
   fd set rfds;
   int result;
   struct timeval tval;
   struct timeval *tptr;

   if (monitor conn == NULL)
      return 0;

   result = wpa ctrl recv(monitor conn, buf, &nread);
   if (result < 0) {
      LOGD("wpa ctrl recv failed: %s\n", strerror(errno));
      return -1;
   }
   buf[nread] = '\0';
   if (result == 0 && nread == 0) {
      /* Fabricate an event to pass up */
      LOGD("Received EOF on supplicant socket\n");
      strncpy(buf, WPA EVENT TERMINATING " - signal 0 received", buflen-1);
      buf[buflen-1] = '\0';
      return strlen(buf);
   }
   if (buf[0] == '<') {
      char *match = strchr(buf, '>');
      if (match != NULL) {
         nread -= (match+1-buf);
         memmove(buf, match+1, nread+1);
      }
   }
   return nread;
}
```

8.2.3　Java FrameWork部分

Wi-Fi 系统的 Java 部分代码实现的目录如下：

```
frameworks/base/wifi/java/android/net/wifi/          // Wi-Fi 服务层的内容
frameworks/base/services/java/com/android/server/    // Wi-Fi 部分的接口
```

Wi-Fi 系统 Java 层的核心是根据 IWifiManger 接口所创建的 Binder 服务器端和客户端，服务器端是 WifiService，客户端是 WifiManger。

编译 IWifiManger.aidl 生成文件 IWifiManger.java，并生成 IWifiManger.Stub(服务器端抽象类)和 IWifiManger.Stub.Proxy(客户端代理实现类)。WifiService 通过继承 IWifiManger.Stub 实现，而客户端通过 getService()函数获取 IWifiManger.Stub.Proxy(即 Service 的代理类)，将其作为参数传递给 WifiManger，供其与 WifiService 通信时使用。

具体结构如图 8-5 所示。

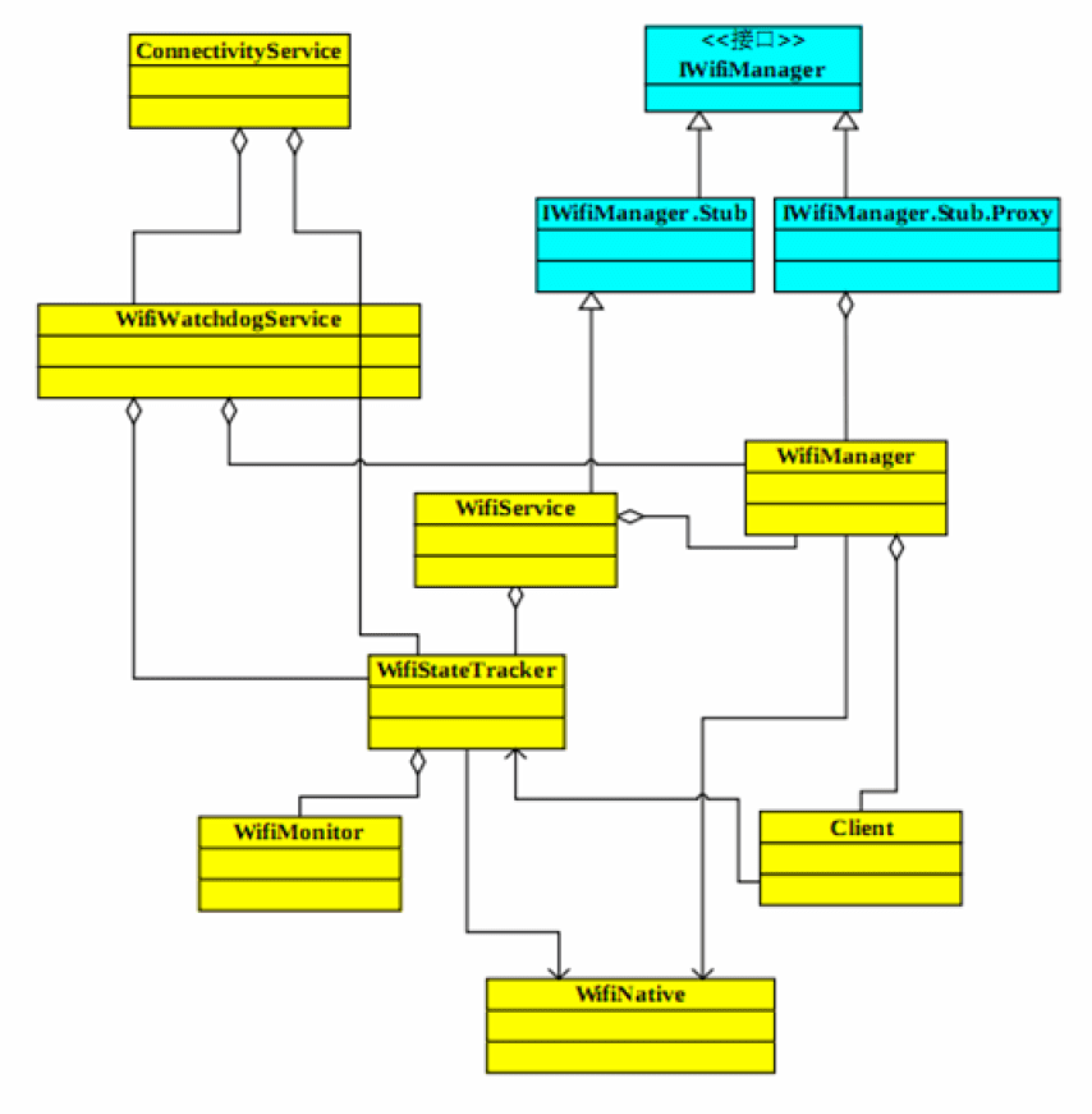


图 8-5　JNI接口结构

图 8-5 中主要构成元素的具体说明如下。

(1) WifiManger 是 Wi-Fi 部分与外界的接口，用户通过它来访问 Wi-Fi 的核心功能。WifiWatchdogService 系统组件也是用 WifiManger 来执行一些具体操作。

(2) WifiService 是服务器端的实现，作为 Wi-Fi 的核心，处理实际的驱动加载、扫描、



链接、断开等命令，以及底层上报的事件。对于主动的命令控制，Wi-Fi 是一个简单的封装，针对来自客户端的控制命令，调用相应的 WifiNative 底层实现。

当接收到客户端的命令后，一般会将其转换成对应的自身消息塞入消息队列中，以便客户端的调用可以及时返回，然后在 WifiHandler 的 handleMessage()中处理对应的消息。而底层上报的事件，WifiService 则通过启动 WifiStateTracker 来负责处理。

(3) WifiStateTracker 和 WifiMonitor 的具体功能如下。

- WifiStateTracker 除了负责 Wi-Fi 的电源管理模式等功能外，其核心是 WifiMonitor 所实现的事件轮询机制，以及消息处理函数 handleMessage()。
- WifiMonitor 通过开启一个 MonitorThread 来实现事件的轮询，轮询的关键函数是前面提到的阻塞式函数 WifiNative.waitForEvent()。获取事件后，WifiMonitor 通过一系列的 Handler 通知给 WifiStateTracker。这里 WifiMonitor 的通知机制是将底层事件转换成 WifiStateTracker 所能识别的消息，塞入 WifiStateTracker 的消息循环中，最终在 handleMessage()中由 WifiStateTracker 完成对应的处理。

WifiStateTracker 同样是 Wi-Fi 部分与外界的接口，它不像 WifiManger 那样直接被实例化来操作，而是通过 Intent 机制来发消息通知给客户端注册的 BroadcastReceiver，以完成和客户端的接口。

(4) WifiWatchdogService 是 ConnectivityService 所启动的服务，但它并不是通过 Binder 来实现的服务。它的作用是监控同一个网络内的接入点(Access Point)，如果当前接入点的 DNS 无法 ping 通，就自动切换到下一个接入点。WifiWatchdogService 通过 WifiManger 和 WifiStateTracker 辅助完成具体的控制动作。在 WifiWatchdogService 初始化时，通过 registerForWifiBroadcasts 注册获取网络变化的 BroadcastReceiver，也就是捕获 WifiStateTracker 所发出的通知消息，并开启一个 WifiWatchdogThread 线程来处理获取的消息。通过更改 Setting.Secure.WIFI_WARCHDOG_ON 的配置，可以开启和关闭 WifiWatchdogService。

8.2.4　Setting中的设置部分

Android 的 Settings 应用程序对 Wi-Fi 的使用，是典型的 Wi-Fi 应用方式，也是用户可见的 Android Wi-Fi 管理程序。此部分源代码的目录如下：

```
packages/apps/Settings/src/com/android/settings/wifi/
```

Setting 里的 Wi-Fi 部分是用户可见的设置界面，提供 Wi-Fi 开关、扫描 AP、链接/断开的基本功能。另外，通过实现 WifiLayer.Callback 接口提供了一组回调函数，用以响应用户关心的 Wi-Fi 状态的变化。

WifiEnabler 和 WifiLayer 都是 WifiSettings 的组成部分，同样通过 WifiManger 来完成实际的功能，也同样注册一个 BroadcastReceiver 来响应 WifiStateTracker 所发出的通知消息。WifiEnabler 其实是一个比较简单的类，提供开启和关闭 Wi-Fi 的功能，设置其中的外层 Wi-Fi 开关菜单，就是直接通过它来做到的；而 WifiLayer 则提供更复杂的一些 Wi-Fi 功能，如 AP 选择等以供用户自定义。

具体结构如图 8-6 所示。

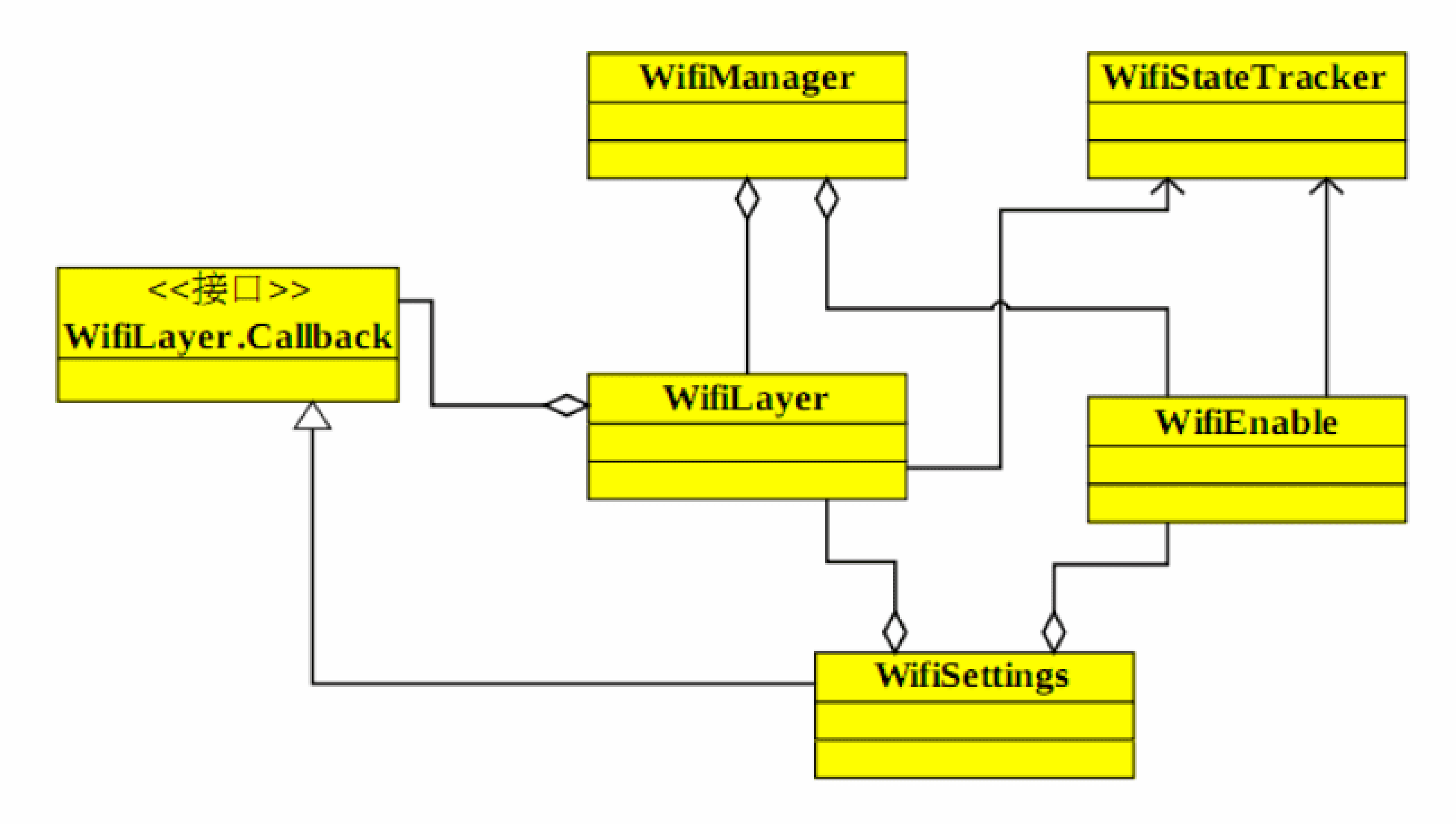


图 8-6　Setting中的Wi-Fi设置结构

8.3　开发 Wi-Fi 应用程序

经过本章前面内容的学习，已经了解了 Android 系统 Wi-Fi 的基本知识。根据对上述从底层到应用的学习，了解了 Wi-Fi 的工作原理和机制。本节将根据前面所学应用到具体实践中，通过具体实例来掌握在 Android 中开发 Wi-Fi 应用的基本知识。

8.3.1　WifiManager类

在应用层开发 Wi-Fi 程序，其实就是使用 WifiManager 类来开发应用程序。在此类中提供了监控 Wi-Fi 状态的方法，主要有以下 5 种状态。

- WifiManager.WIFI_STATE_ENABLING：表示 Wi-Fi 已经打开。
- WifiManager.WIFI_STATE_DISABLING：表示 Wi-Fi 正在关闭而无法关闭。
- WifiManager.WIFI_STATE_DISABLED：表示 Wi-Fi 已经关闭。
- WifiManager.WIFI_STATE_ENABLED：表示 Wi-Fi 已经打开无法再打开。
- WifiManager.WIFI_STATE_UNKNOWN：表示 Wi-Fi 无法识别。

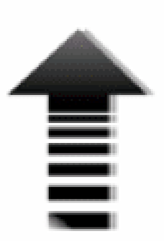
在具体实现上，我们先定义一个复选框 CheckBox，然后捕捉 CheckBox 的点击事件，根据对应的状态显示对应的提示。例如可以用下面的代码检测 Wi-Fi 是否启动：

```
WifiManager wm = (WifiManager)
context.getSystemService(Context.WIFI_SERVICE);
if(wm.getWifiState() == WifiManager.WIFI_STATE_ENABLED){
   return true;
}
```

设置 Wi-Fi 可用的代码如下：

```
wifimanager.setWifiEnabled(!wifiEnabled);
```





例如下面的一段代码是通用的 Wi-Fi 应用程序。

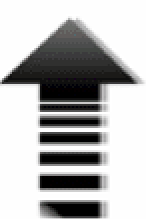
```
import java.util.List;

import android.content.Context;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiConfiguration;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.net.wifi.WifiManager.WifiLock;

public class WifiAdmin
{
    //定义 WifiManager 对象
    private WifiManager mWifiManager;
    //定义 WifiInfo 对象
    private WifiInfo mWifiInfo;
    //扫描出的网络连接列表
    private List<ScanResult> mWifiList;
    //网络连接列表
    private List<WifiConfiguration> mWifiConfiguration;
    //定义一个 WifiLock
    WifiLock mWifiLock;
    //构造器
    public  WifiAdmin(Context context)
    {
        //取得 WifiManager 对象
        mWifiManager = (WifiManager) context.getSystemService
          (Context.WIFI SERVICE);
        //取得 WifiInfo 对象
        mWifiInfo = mWifiManager.getConnectionInfo();
    }
    //打开 Wi-Fi
    public void OpenWifi()
    {
        if (!mWifiManager.isWifiEnabled())
        {
            mWifiManager.setWifiEnabled(true);

        }
    }
    //关闭 Wi-Fi
    public void CloseWifi()
    {
        if (!mWifiManager.isWifiEnabled())
        {
            mWifiManager.setWifiEnabled(false);
        }
    }
    //锁定 WifiLock，当下载大文件时需要锁定
    public void AcquireWifiLock()
```

```
{
    mWifiLock.acquire();
}
//解锁 WifiLock
public void ReleaseWifiLock()
{
    //判断时候锁定
    if (mWifiLock.isHeld())
    {
        mWifiLock.acquire();
    }
}
//创建一个 WifiLock
public void CreatWifiLock()
{
    mWifiLock = mWifiManager.createWifiLock("Test");
}
//得到配置好的网络
public List<WifiConfiguration> GetConfiguration()
{
    return mWifiConfiguration;
}
//指定配置好的网络进行连接
public void ConnectConfiguration(int index)
{
    //索引，大于配置好的网络索引返回
    if(index > mWifiConfiguration.size())
    {
        return;
    }
    //连接配置好的指定 ID 的网络
    mWifiManager.enableNetwork(mWifiConfiguration.get(index).networkId,
      true);
}
public void StartScan()
{
    mWifiManager.startScan();
    //得到扫描结果
    mWifiList = mWifiManager.getScanResults();
    //得到配置好的网络连接
    mWifiConfiguration = mWifiManager.getConfiguredNetworks();
}
//得到网络列表
public List<ScanResult> GetWifiList()
{
    return mWifiList;
}
//查看扫描结果
public StringBuilder LookUpScan()
{
    StringBuilder stringBuilder = new StringBuilder();
```



```
        for (int i = 0; i < mWifiList.size(); i++)
        {
            stringBuilder.append("Index "+new Integer(i + 1).toString() + ":");
            //将 ScanResult 信息转换成一个字符串包
            //其中包括：BSSID、SSID、capabilities、frequency、level
            stringBuilder.append((mWifiList.get(i)).toString());
            stringBuilder.append("\n");
        }
        return stringBuilder;
    }
    //得到 MAC 地址
    public String GetMacAddress()
    {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.getMacAddress();
    }
    //得到接入点的 BSSID
    public String GetBSSID()
    {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.getBSSID();
    }
    //得到 IP 地址
    public int GetIPAddress()
    {
        return (mWifiInfo == null) ? 0 : mWifiInfo.getIpAddress();
    }
    //得到连接的 ID
    public int GetNetworkId()
    {
        return (mWifiInfo == null) ? 0 : mWifiInfo.getNetworkId();
    }
    //得到 WifiInfo 的所有信息包
    public String GetWifiInfo()
    {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.toString();
    }
    //添加一个网络并连接
    public void AddNetwork(WifiConfiguration wcg)
    {
        int wcgID = mWifiManager.addNetwork(wcg);
        mWifiManager.enableNetwork(wcgID, true);
    }
    //断开指定 ID 的网络
    public void DisconnectWifi(int netId)
    {
        mWifiManager.disableNetwork(netId);
        mWifiManager.disconnect();
    }
}
```

在具体开发 Wi-Fi 应用程序时需要注意两点。

第一是需要检测当前设备是否有可用的 Wi-Fi。例如下面的检测代码：

```
mWifiManager = (WifiManager)
context.getSystemService(Context.WIFI_SERVICE);
if (mWifiManager != null) {
            List<ScanResult> wifiScanResults = mWifiManager.getScanResults();
            if (wifiScanResults != null && wifiScanResults.size() != 0) {

            }
         }
```

第二点是需要在程序中声明一些相关的权限。Wi-Fi 的主要操作权限有以下四个。

- CHANGE_NETWORK_STATE：允许修改网络状态的权限。
- CHANGE_WIFI_STATE：允许修改 Wi-Fi 状态的权限。
- ACCESS_NETWORK_STATE：允许访问网络状态的权限。
- ACCESS_WIFI_STATE：允许访问 Wi-Fi 状态的权限。

例如下面的代码：

```
<uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_CHECKIN_PROPERTIES">
        </uses-permission>
    <uses-permission android:name="android.permission.WAKE_LOCK">
        </uses-permission>
    <uses-permission android:name="android.permission.INTERNET">
        </uses-permission>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE">
        </uses-permission>
    <uses-permission android:name="android.permission.MODIFY_PHONE_STATE">
        </uses-permission>
```

8.3.2 在Android系统中控制Wi-Fi

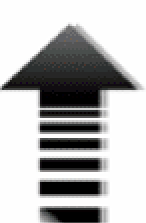
了解了 Wi-Fi 的基本知识后，在接下来的内容中，将通过两个具体实例的实现过程，来讲解在 Android 系统中开发 Wi-Fi 应用程序的基本流程。

实 例	功 能	源码路径
实例 8-1	在 Android 系统中控制 Wi-Fi	下载路径:\daima\8\control

本实例的功能是在 Android 系统中控制 Wi-Fi 的状态，具体实现流程如下。

(1) 编写布局文件 main.xml，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:background="@drawable/white"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
```



```
  <TextView
    android:id="@+id/myTextView1"
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:textColor="@drawable/blue"
    android:text="@string/hello"
  />
  <CheckBox
    android:id="@+id/myCheckBox1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="@string/str checked"
    android:textColor="@drawable/blue"
  />
</LinearLayout>
```

(2) 实现主程序文件 control.java，具体实现流程如下。

① 创建 WifiManager 对象 mWiFiManager01，具体代码如下：

```
public class control extends Activity
{
  private TextView mTextView01;
  private CheckBox mCheckBox01;

  /* 创建 WifiManager 对象 */
  private WifiManager mWiFiManager01;
```

② 定义 mTextView01 和 mCheckBox01，分别用于显示提示文本和获取复选框的选择状态。具体代码如下：

```
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mTextView01 = (TextView) findViewById(R.id.myTextView1);
    mCheckBox01 = (CheckBox) findViewById(R.id.myCheckBox1);
```

③ 以 getSystemService 取得 WIFI_SERVICE，具体代码如下：

```
    mWiFiManager01 = (WifiManager)
      this.getSystemService(Context.WIFI_SERVICE);
```

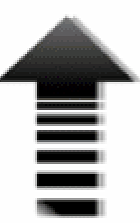
④ 通过 if 语句来判断运行程序后的 Wi-Fi 状态是否打开或打开中，这样便可显示对应的提示信息。具体代码如下：

```
    /* 判断运行程序后的 Wi-Fi 状态是否打开或打开中 */
    if(mWiFiManager01.isWifiEnabled())
    {
      /* 判断 Wi-Fi 状态是否“已打开” */
      if(mWiFiManager01.getWifiState()==
```

```
      WifiManager.WIFI_STATE_ENABLED)
    {
      /* 若 Wi-Fi 已打开，将选取项打勾 */
      mCheckBox01.setChecked(true);
      /* 更改选取项文字为关闭 Wi-Fi*/
      mCheckBox01.setText(R.string.str_uncheck);
    }
    else
    {
      /* 若 Wi-Fi 未打开，将选取项勾取消 */
      mCheckBox01.setChecked(false);
      /* 更改选取项文字为打开 Wi-Fi*/
      mCheckBox01.setText(R.string.str_checked);
    }
  }
  else
  {
    mCheckBox01.setChecked(false);
    mCheckBox01.setText(R.string.str_checked);
  }
```

⑤ 通过 mCheckBox01.setOnClickListener 来捕捉 CheckBox 的点击事件，用 onClick(View v)方法获取用户的点击，然后根据 if 语句的操作需求来执行对应操作，并根据需要输出对应的提示信息。具体代码如下：

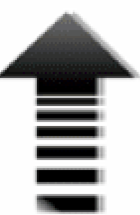
```
  mCheckBox01.setOnClickListener(
  new CheckBox.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      // TODO Auto-generated method stub
      /* 当选取项为取消选取状态 */
      if(mCheckBox01.isChecked()==false)
      {
        /* 尝试关闭 Wi-Fi 服务 */
        try
        {
          /* 判断 Wi-Fi 状态是否为已打开 */
          if(mWiFiManager01.isWifiEnabled() )
          {
            /* 关闭 Wi-Fi */
            if(mWiFiManager01.setWifiEnabled(false))
            {
              mTextView01.setText(R.string.str_stop_wifi_done);
            }
            else
            {
              mTextView01.setText(R.string.str_stop_wifi_failed);
            }
          }
```



```
        else
        {
          /* Wi-Fi 状态不为已打开状态时 */
          switch(mWiFiManager01.getWifiState())
          {
            /* Wi-Fi 正在打开过程中，导致无法关闭... */
            case WifiManager.WIFI_STATE_ENABLING:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str_stop_wifi_failed)+":"+
                getResources().getText
                (R.string.str_wifi_enabling)
              );
              break;
            /* Wi-Fi 正在关闭过程中，导致无法关闭... */
            case WifiManager.WIFI_STATE_DISABLING:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str_stop_wifi_failed)+":"+
                getResources().getText
                (R.string.str_wifi_disabling)
              );
              break;
            /* Wi-Fi 已经关闭 */
            case WifiManager.WIFI_STATE_DISABLED:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str_stop_wifi_failed)+":"+
                getResources().getText
                (R.string.str_wifi_disabled)
              );
              break;
            /* 无法取得或辨识 Wi-Fi 状态 */
            case WifiManager.WIFI_STATE_UNKNOWN:
            default:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str_stop_wifi_failed)+":"+
                getResources().getText
                (R.string.str_wifi_unknow)
              );
              break;
          }
          mCheckBox01.setText(R.string.str_checked);
        }
      }
```




```
    catch (Exception e)
    {
      Log.i("HIPPO", e.toString());
      e.printStackTrace();
    }
  }
  else if(mCheckBox01.isChecked()==true)
  {
    /* 尝试打开 Wi-Fi 服务 */
    try
    {
      /* 确认 Wi-Fi 服务是关闭且不在打开作业中 */
      if(!mWiFiManager01.isWifiEnabled() &&
         mWiFiManager01.getWifiState()!=
         WifiManager.WIFI STATE ENABLING )
      {
        if(mWiFiManager01.setWifiEnabled(true))
        {
          switch(mWiFiManager01.getWifiState())
          {
            /* Wi-Fi 正在打开过程中，导致无法打开... */
            case WifiManager.WIFI STATE ENABLING:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str wifi enabling)
              );
              break;
            /* Wi-Fi 已经为打开，无法再次打开... */
            case WifiManager.WIFI STATE ENABLED:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str start wifi done)
              );
              break;
            /* 其他未知的错误 */
            default:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str start wifi failed)+":"+
                getResources().getText
                (R.string.str wifi unknow)
              );
              break;
          }
        }
        else
        {
          mTextView01.setText(R.string.str_start_wifi_failed);
```



```
            }
          }
          else
          {
            switch(mWiFiManager01.getWifiState())
            {
              /* Wi-Fi 正在打开过程中，导致无法打开... */
              case WifiManager.WIFI_STATE_ENABLING:
                mTextView01.setText
                (
                  getResources().getText
                  (R.string.str_start_wifi_failed)+":"+
                  getResources().getText
                  (R.string.str_wifi_enabling)
                );
                break;
              /* Wi-Fi 正在关闭过程中，导致无法打开... */
              case WifiManager.WIFI_STATE_DISABLING:
                mTextView01.setText
                (
                  getResources().getText
                  (R.string.str_start_wifi_failed)+":"+
                  getResources().getText
                  (R.string.str_wifi_disabling)
                );
                break;
              /* Wi-Fi 已经关闭 */
              case WifiManager.WIFI_STATE_DISABLED:
                mTextView01.setText
                (
                  getResources().getText
                  (R.string.str_start_wifi_failed)+":"+
                  getResources().getText
                  (R.string.str_wifi_disabled)
                );
                break;
              /* 无法取得或识别 Wi-Fi 状态 */
              case WifiManager.WIFI_STATE_UNKNOWN:
              default:
                mTextView01.setText
                (
                  getResources().getText
                  (R.string.str_start_wifi_failed)+":"+
                  getResources().getText
                  (R.string.str_wifi_unknow)
                );
                break;
            }
          }
          mCheckBox01.setText(R.string.str_uncheck);
        }
```



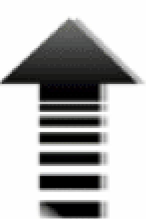

```
        catch (Exception e)
        {
          Log.i("HIPPO", e.toString());
          e.printStackTrace();
        }
      }
    }
  });
}
```

⑥ 定义 mMakeTextToast(String str, boolean isLong)，用于根据当前操作显示对应的提示性信息。具体代码如下：

```
public void mMakeTextToast(String str, boolean isLong)
{
  if(isLong==true)
  {
    Toast.makeText(example110.this, str, Toast.LENGTH LONG).show();
  }
  else
  {
    Toast.makeText(example110.this, str, Toast.LENGTH SHORT).show();
  }
}

@Override
protected void onResume()
{
  // TODO Auto-generated method stub

  /* 在 onResume 重写事件为取得打开程序当下 Wi-Fi 的状态 */
  try
  {
    switch(mWiFiManager01.getWifiState())
    {
      /* Wi-Fi 已经在打开状态... */
      case WifiManager.WIFI STATE ENABLED:
        mTextView01.setText
        (
          getResources().getText(R.string.str wifi enabling)
        );
        break;
      /* Wi-Fi 正在打开过程中... */
      case WifiManager.WIFI STATE ENABLING:
        mTextView01.setText
        (
          getResources().getText(R.string.str wifi enabling)
        );
        break;
      /* Wi-Fi 正在关闭过程中... */
      case WifiManager.WIFI_STATE_DISABLING:
```



```
          mTextView01.setText
          (
            getResources().getText(R.string.str wifi disabling)
          );
          break;
        /* Wi-Fi 已经关闭 */
        case WifiManager.WIFI STATE DISABLED:
          mTextView01.setText
          (
            getResources().getText(R.string.str wifi disabled)
          );
          break;
        /* 无法取得或识别 Wi-Fi 状态 */
        case WifiManager.WIFI STATE UNKNOWN:
        default:
          mTextView01.setText
          (
            getResources().getText(R.string.str wifi unknow)
          );
          break;
      }
    }
    catch(Exception e)
    {
      mTextView01.setText(e.toString());
      e.getStackTrace();
    }
    super.onResume();
  }

  @Override
  protected void onPause()
  {
    // TODO Auto-generated method stub
    super.onPause();
  }
}
```

(3) 编写文件 strings.xml，此处设置了在屏幕中显示的文本内容。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello"></string>
  <string name="app name"></string>
  <string name="str_checked">打开....</string>
  <string name="str_uncheck">关闭....</string>
  <string name="str_start_wifi_failed">打开失败</string>
  <string name="str_start_wifi_done">打开成功</string>
  <string name="str_stop_wifi_failed">打开失败</string>
  <string name="str stop wifi done">关闭成功</string>
  <string name="str_wifi_enabling">正在启动....</string>
```

```
  <string name="str_wifi_disabling">正在关闭....</string>
  <string name="str_wifi_disabled">已关闭</string>
  <string name="str_wifi_unknow">未知....</string>
</resources>
```

(4) 在文件 AndroidManifest.xml 中添加对 Wi-Fi 的访问以及对网络状态的权限。具体代码如下：

```
  <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.WAKE_LOCK" />
```

到此为止，整个实例介绍完毕，执行后会显示两个按钮，如图 8-7 所示。当选择复选框后会执行对应的操作处理，并且显示对应的提示信息。



图 8-7　执行效果

在此需要说明的是，由于 Android 模拟器不支持 Wi-Fi 和蓝牙，所以执行上述程序时返回的网卡状态都是 WIFI_STATE_UNKNOWN，即表示网卡未知的状态。

8.3.3　在Android系统中打开或关闭Wi-Fi网卡

实　例	功　能	源码路径
实例 8-2	控制 Wi-Fi	下载路径:\daima\8\Android_Wifi

本实例新建了一个 Android 应用程序，在 main.xml 中添加三个按钮，点击这三个按钮分别可以打开 Wi-Fi 网卡、关闭 Wi-Fi 网卡、检查网卡的当前状态。本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
```

```
<Button
    android:id="@+id/startButton"
    android:layout width="300dp"
    android:layout height="wrap content"
    android:text="打开 WIFI 网卡"
   />
<Button
    android:id="@+id/stopButton"
    android:layout width="300dp"
    android:layout height="wrap content"
    android:text="关闭 WIFI 网卡"
   />
<Button
    android:id="@+id/checkButton"
    android:layout width="300dp"
    android:layout height="wrap content"
    android:text="检查 WIFI 网卡状态"
   />
</LinearLayout>
```

(2) 编写主程序文件 Android_Wifi.java，具体代码如下：

```
package idea.org;

import android.app.Activity;
import android.content.Context;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class Android Wifi extends Activity {
    private Button startButton=null;
    private Button stopButton=null;
    private Button checkButton=null;
    WifiManager wifiManager=null;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      startButton=(Button)findViewById(R.id.startButton);
      stopButton=(Button)findViewById(R.id.stopButton);
      checkButton=(Button)findViewById(R.id.checkButton);
      startButton.setOnClickListener(new startButtonListener());
      stopButton.setOnClickListener(new stopButtonListener());
      checkButton.setOnClickListener(new checkButtonListener());
   }
   class startButtonListener implements OnClickListener
   {
       /* (non-Javadoc)
```




```
         * @see android.view.View.OnClickListener#onClick(android.view.View)
          */
         @Override
         public void onClick(View v) {
             // TODO Auto-generated method stub
             wifiManager=(WifiManager)Android_Wifi.this.getSystemService
                (Context.WIFI_SERVICE);
             wifiManager.setWifiEnabled(true);
             System.out.println("wifi state --->"+wifiManager.getWifiState());
             Toast.makeText(Android_Wifi.this, "当前网卡状态为: "+
                 wifiManager.getWifiState(), Toast.LENGTH_SHORT).show();
         }

    }
    class stopButtonListener implements OnClickListener
    {
         /* (non-Javadoc)
          * @see android.view.View.OnClickListener#onClick(android.view.View)
          */
         @Override
         public void onClick(View v) {
             // TODO Auto-generated method stub
    wifiManager=(WifiManager)Android_Wifi.this.getSystemService
         (Context.WIFI_SERVICE);
    wifiManager.setWifiEnabled(false);
    System.out.println("wifi state --->"+wifiManager.getWifiState());
    Toast.makeText(Android_Wifi.this, "当前网卡状态为: "+wifiManager.getWifiState(),
       Toast.LENGTH_SHORT).show();
         }

    }
    class checkButtonListener implements OnClickListener
    {
         /* (non-Javadoc)
         * @see android.view.View.OnClickListener#onClick(android.view.View)
          */
         @Override
         public void onClick(View v) {
             // TODO Auto-generated method stub
             wifiManager=(WifiManager)Android_Wifi.this.getSystemService
                (Context.WIFI_SERVICE);
             System.out.println("wifi state --->"+wifiManager.getWifiState());
             Toast.makeText(Android_Wifi.this, "当前网卡状态为: "+
               wifiManager.getWifiState(), Toast.LENGTH_SHORT).show();
         }
    }
}
```

(3) 在文件 AndroidManifest.xml 中声明 Wi-Fi 权限，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
```



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="idea.org"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="11" />
   <application android:icon="@drawable/icon" android:label="@string/app name">
      <activity android:name=".Android Wifi"
              android:label="@string/app name">
          <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
      </activity>
   </application>
   <uses-permission android:name="android.permission.CHANGE NETWORK STATE"/>
   <uses-permission android:name="android.permission.CHANGE WIFI STATE"/>
   <uses-permission android:name="android.permission.ACCESS NETWORK STATE"/>
   <uses-permission android:name="android.permission.ACCESS WIFI STATE"/>
</manifest>
```

执行后的效果如图 8-8 所示。



图 8-8　执行效果

依次单击“打开 WIFI 网卡”、“关闭 WIFI 网卡”、“检查 WIFI 网卡状态”三个按钮，控制台输出对应内容。在 Eclipse 中会显示对应的状态值，如图 8-9 所示。

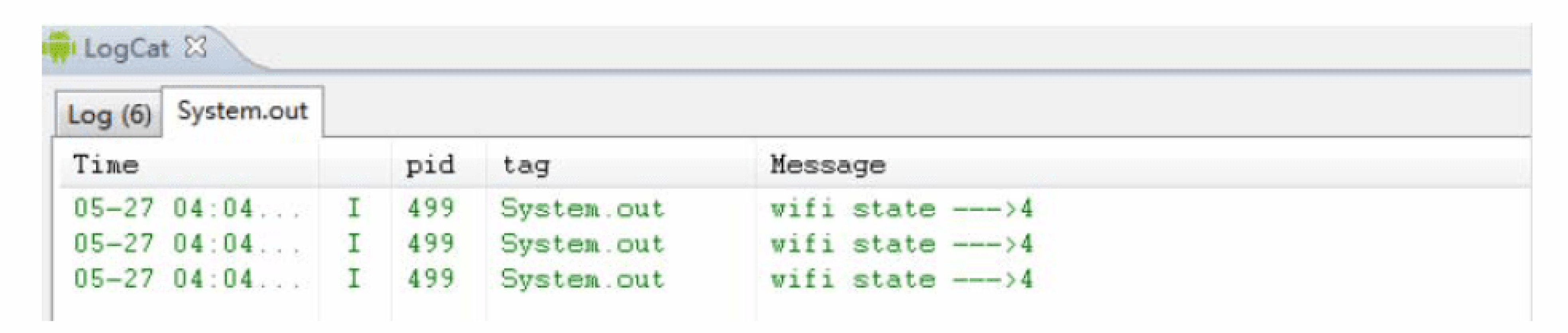


图 8-9　Eclipse中的值

第9章

在Android中开发RSS应用

RSS 是在线共享内容的一种简易方式(也叫聚合内容，Really Simple Syndication)。通常在时效性比较强的内容上使用 RSS 订阅能更快速地获取信息。网站提供 RSS 输出，有利于让用户获取网站内容的最新更新。本章将讲解在 Android 手机中实现 RSS 处理的基本知识。

9.1 RSS 基础

RSS 通常被用在时效性比较强的内容上，通过使用 RSS 订阅可以更快速地获取信息。在网站中提供 RSS 输出，有利于让用户获取网站内容的最新更新。

9.1.1 RSS的用途

RSS 的主要用途如下。

(1) 订阅 Blog(你可以订阅工作中所需的技术文章；也可以订阅与你有共同爱好的作者的 Blog，总之，你对什么感兴趣就可以订阅了什么)。

(2) 订阅新闻(无论是奇闻怪事、明星消息、体坛风云，只要你想知道的，都可以订阅)。

(3) 再也不用一个网站一个网站、一个网页一个网页去逛了。只要将你需要的内容订阅在一个 RSS 阅读器中，这些内容就会自动出现在你的阅读器中，你也不必为了一个急切想知道的消息而不断地刷新网页，因为一旦有了更新，RSS 阅读器就会通知你。

其实订阅 RSS 新闻内容要先安装一个 RSS 阅读器，然后将提供 RSS 服务的网站加入到 RSS 阅读器的频道即可。具体如下。

(1) 选择有价值的 RSS 信息源。

(2) 启动 RSS 订阅程序，将信息源添加到自己的 RSS 阅读器或者在线 RSS。

(3) 接收并获取定制的 RSS 信息。

9.1.2 RSS阅读器

RSS 阅读器可以分为以下三类。

1) 计算机桌面程序

大多数阅读器是运行在计算机桌面上的应用程序，通过所订阅网站的新闻供应，可自动、定时地更新新闻标题。在该类阅读器中，有 Awasu、FeedDemon 和 RSSReader 三款流行的阅读器，均提供免费试用版和付费高级版。

2) 新闻阅读器

新闻阅读器通常内嵌于已在计算机中运行的应用程序中。例如，NewsGator 内嵌在微软的 Outlook 中，所订阅的新闻标题位于 Outlook 的收件箱文件夹中。另外，Pluck 内嵌在 Internet Explorer 浏览器中。

3) 在线的 Web RSS 阅读器

在线 Web RSS 阅读器的优势是，不需要安装任何软件就可以获得 RSS 阅读的便利，并且可以保存阅读状态，推荐和收藏自己感兴趣的文章。提供此服务的网站有两类，一类是专门提供 RSS 阅读器的网站，例如国外的 Google Reader，国内的鲜果、抓虾；另一类是提供个性化首页的网站，例如国外的 Netvibes、Pageflakes，国内的雅蛙、阔地。



9.1.3　RSS的语法

RSS 2.0 的语法规则非常简单并十分的严格，看下面的代码：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
<channel>

<title>W3Schools</title>
<link>http://www.w3schools.com</link>
<description>W3Schools Web Tutorials </description>

<item>
<title>RSS Tutorial</title>
<link>http://www.w3schools.com/rss</link>
<description>Check out the RSS tutorial
on W3Schools.com</description>
</item>

</channel>
</rss>
```

其中，<channel>元素内是描述 RSS feed 的地方。

RSS 的<channel>元素是项目内容显示的地方。它就像 RSS 的标题。通常它不会频繁地改动。有三个内部元素是必选的，分别是<title>、<link>和<description>。具体说明如下。

- <title>：包含你的网站和你的 RSS feed 简短说明。
- <link>：定义你的网站主页的链接。
- <description>：描述你的 RSS feed。

<channel>内的可选元素如下。

- <category>：定义一个或多个频道分类。
- <cloud>：允许更新通告。
- <copyright>：提醒有关版权。
- <docs>：频道所使用的 RSS 版本文档 URL。
- <generator>：如果频道是自动生成器产生的，就在这里定义。
- <image>：为频道加图片。
- <language>：描述频道所使用的语言。
- <lastBuildDate>：定义频道最近一次改动的时间。
- <managingEditor>：定义编辑站点人员的 E-mail 地址。
- <pubDate>：定义频道最新的发布时间。
- <rating>：页面评估。
- <ttl>：存活的有效时间。
- <webMaster>：定义站点的邮件地址。

<item>元素内是你的网站连接和描述更新内容的地方。<item>是显示 RSS 更新内容的地方。它像是文章的标题。当你的站点有更新时，RSS feed 中的<item>元素就会被建立起

来。<item>元素中有几个可选的元素，但<title>和<description>是必选元素。

一个 RSS 的<item>应该包括：<title>、<link>和<description>元素。

- <title>：项目的题目，应该用十分简短的描述。
- <link>：项目所关联的链接。
- <description>：RSS feed 的描述部分，这应该是描述你的 RSS feed 项目的。

<item>中的可选元素如下。

- <author>：定义作者。
- <category>：类别。
- <comments>：针对项目的评论页 URL。
- <enclosure>：描述一个与项目有关的媒体对象。
- <guid>：针对项目定义独特的标志。
- <pubDate>：项目发布时间。
- <source>：转载地址(源地址)。

在<description>中建议使用<![CDATA[ ]]>，所有在 CDATA 部件之间的文本都会被解析器忽略。

注意：CDATA 部件之间不能再包含 CDATA 部件(不能嵌套)。如果 CDATA 部件包含了字符"]]>"或者 CDATA，将很有可能出错。同样要注意在字符串"]]>"之间没有空格或者换行符。

9.2 SAX 介绍

SAX 是 Simple API for XML 的缩写，不但是指一种接口，而且也是指一个软件包。SAX 最初是由 David Megginson 采用 Java 语言开发的，之后 SAX 很快在 Java 开发者中流行起来。Sun 现在负责管理其原始 API 的开发工作，这是一种公开的、开放源代码软件。不同于其他大多数 XML 标准的是，SAX 没有语言开发商必须遵守的标准 SAX 参考版本。因此，SAX 的不同实现可能采用区别很大的接口。本节简要介绍 SAX 技术的基本知识。

9.2.1 SAX的原理

作为接口，SAX 是事件驱动型 XML 解析的一个标准接口(Standard Interface)，不会改变，已被 OASIS(Organization for the Advancement of Structured Information Standards)所采纳。作为软件包，SAX 最早的开发始于 1997 年 12 月，由一些在互联网上分散的程序员合作进行。后来，参与开发的程序员越来越多，组成了互联网上的 XML-DEV 社区。五个月以后，1998 年 5 月，SAX 1.0 版由 XML-DEV 正式发布。目前，最新的版本是 SAX 2.0。2.0 版本在多处与 1.0 版本不兼容，包括一些类和方法的名字。

SAX 的工作原理简单地说就是对文档进行顺序扫描，当扫描到文档(document)开始与结束、元素(element)开始与结束等地方时通知事件处理函数，由事件处理函数做相应动



作，然后继续同样的扫描，直至文档结束。

大多数 SAX 实现都会产生以下五种类型的事件。

- 在文档的开始和结束时触发文档处理事件。
- 在文档内每一 XML 元素接受解析的前后触发元素事件。
- 任何元数据通常都由单独的事件交付。
- 在处理文档的 DTD 或 Schema 时产生 DTD 或 Schema 事件。
- 产生错误事件用来通知主机应用程序解析错误。

9.2.2　基于对象和基于事件的接口

语法分析器有两类接口：基于对象的接口和基于事件的接口。

DOM 是基于对象的语法分析器的标准的 API。作为基于对象的接口，DOM 通过在内存中显式地构建对象树来与应用程序通信。对象树是 XML 文件中元素树的精确映射。

DOM 易于学习和使用，因为它与基本 XML 文档紧密匹配。特别以 XML 为中心的应用程序(例如，浏览器和编辑器)也是很理想的。以 XML 为中心的应用程序为了操纵 XML 文档而操纵 XML 文档。

然而，对于大多数应用程序，处理 XML 文档只是其众多任务中的一种。例如，记账软件包可能导入 XML 发票，但这不是其主要活动。计算账户余额、跟踪支出以及使付款与发票匹配才是主要活动。记账软件包可能已经具有一个数据结构(最有可能是数据库)。DOM 模型不太适合记账应用程序，因为在那种情况下，应用程序必须在内存中维护数据的两份副本(一个是 DOM 树，另一个是应用程序自己的结构)。至少，在内存中维护两次数据会使效率下降。对于桌面应用程序来说，这可能不是主要问题，但是有可能导致服务器瘫痪。对于不是以 XML 为中心的应用程序，SAX 是明智的选择。实际上，SAX 并不在内存中显式地构建文档树。它使应用程序能用最有效率的方法存储数据。

图 9-1 说明了应用程序是如何在 XML 树及其自身数据结构之间进行映射的。

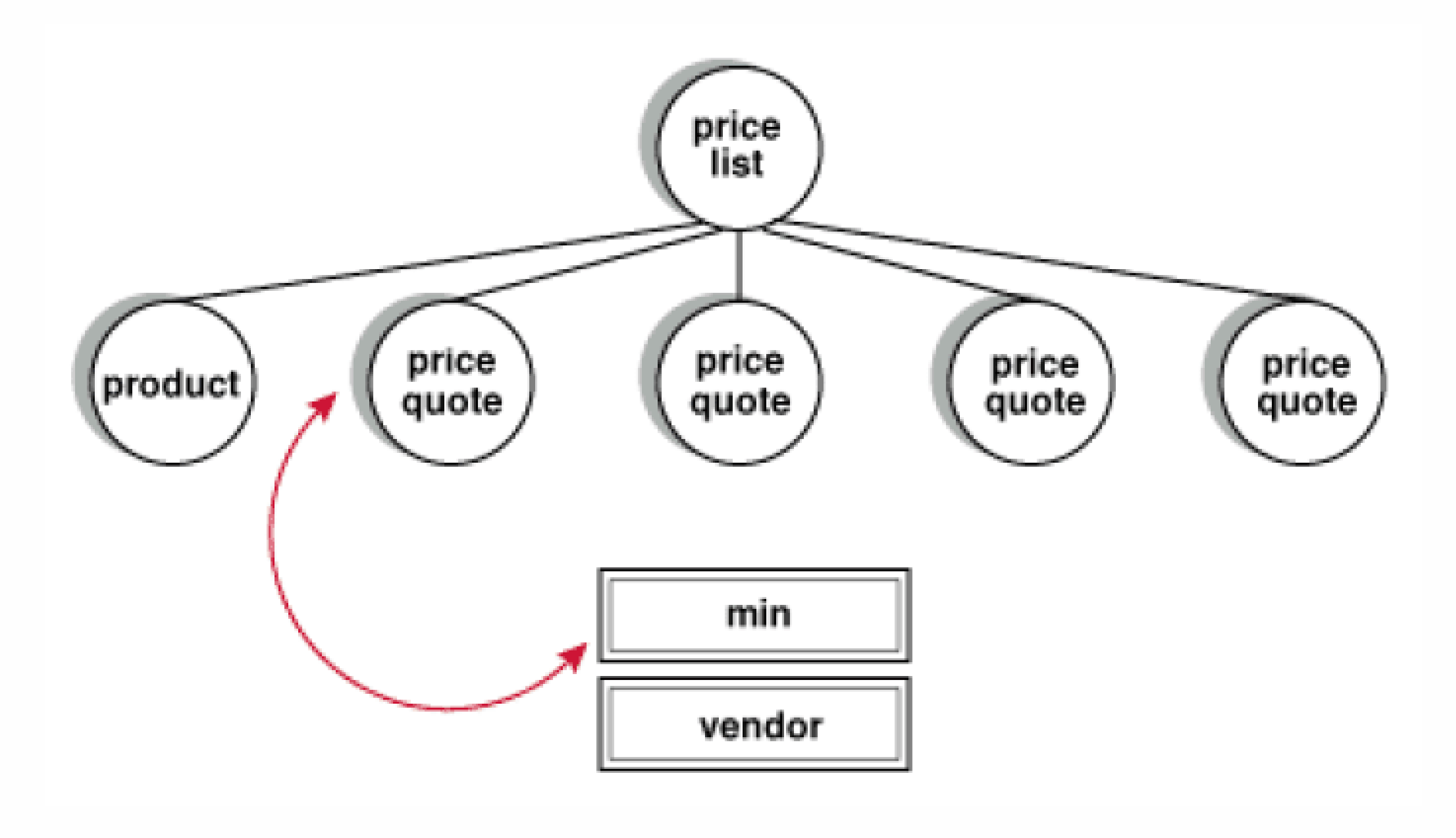


图 9-1　将XML结构映射成应用程序结构

SAX 是基于事件的接口，正如其名称所暗示，基于事件的语法分析器将事件发送给应用程序。这些事件类似于用户界面事件，例如，浏览器中的 onClick 事件或者 Java 中的 AWT/Swing 事件。

事件通知应用程序发生了某件事并需要应用程序做出反应。在浏览器中，通常为响应用户操作而生成事件：当用户单击按钮时，按钮产生一个 onClick 事件。

在 XML 语法分析器中，事件与用户操作无关，而与正在读取的 XML 文档中的元素有关。有对于以下方面的事件。

- 元素开始和结束标记
- 元素内容
- 实体
- 语法分析错误

图 9-2 显示了语法分析器在读取文档时是如何生成事件的。

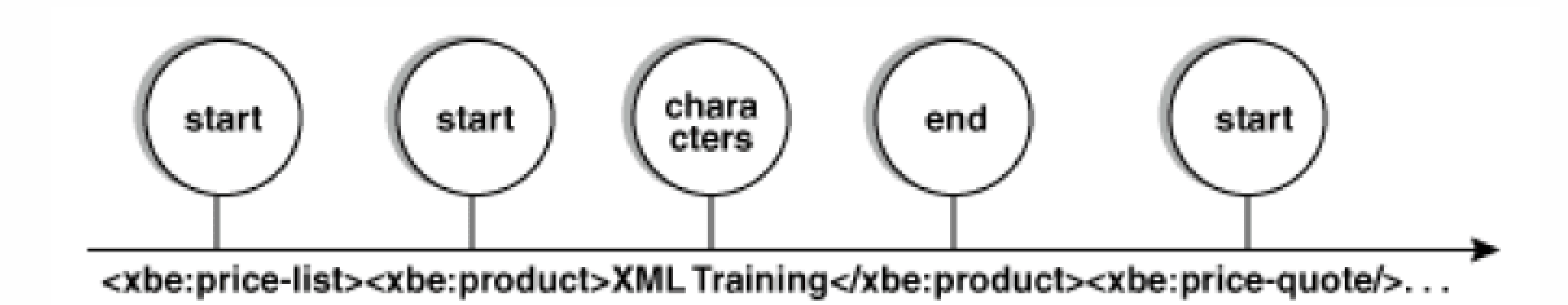


图 9-2　语法分析器生成事件

读者在此可能会问：为什么使用基于事件的接口？

这两种 API 中没有一种在本质上更好；它们适用于不同的需求。经验法则是在需要更多控制时使用 SAX；要增加方便性时，则使用 DOM。例如，DOM 在脚本语言中很流行。

采用 SAX 的主要原因是效率。SAX 比 DOM 做的事要少，但提供了对语法分析器的更多控制。当然，如果语法分析器的工作减少，则意味着您(开发者)有更多的工作要做。而且，正如我们已讨论的，SAX 比 DOM 消耗的资源要少，这是因为它不需要构建文档树。在 XML 早期，DOM 得益于 W3C 批准的官方 API 这一身份。逐渐地，开发者选择了功能性而放弃了方便性，并转向了 SAX。

SAX 的主要限制是它无法向后浏览文档。实际上，激发一个事件后，语法分析器就将其忘记。如您将看到的，应用程序必须显式地缓冲其感兴趣的事件。

9.2.3　常用的接口和类

SAX 将其事件分为以下几个接口。

- ContentHandler：定义与文档本身关联的事件(例如，开始和结束标记)。大多数应用程序都注册这些事件。
- DTDHandler：定义与 DTD 关联的事件。然而，它不定义足够的事件来完整地报告 DTD。如果需要对 DTD 进行语法分析，请使用可选的 DeclHandler。DeclHandler 是 SAX 的扩展，并且不是所有的语法分析器都支持它。
- EntityResolver：定义与装入实体关联的事件。只有少数几个应用程序注册这些事件。
- ErrorHandler：定义错误事件。许多应用程序注册这些事件以便用它们自己的方式报错。

为简化工作，SAX 在 DefaultHandler 类中提供了这些接口的默认实现。在大多数情况下，为应用程序扩展 DefaultHandler 并覆盖相关的方法要比直接实现一个接口更容易。



1. XMLReader

为注册事件处理器并启动语法分析器，应用程序使用 XMLReader 接口。如我们所见，parse()这种 XMLReader 方法，启动语法分析：

```
parser.parse(args[0]);
```

XMLReader 的主要方法如下。

(1) parse()：对 XML 文档进行语法分析。parse()有两个版本：一个接受文件名或 URL，另一个接受 InputSource 对象。

(2) setContentHandler()、setDTDHandler()、setEntityResolver()和 setErrorHandler()：让应用程序注册事件处理器。

(3) setFeature()和 setProperty()：控制语法分析器如何工作。它们采用一个特性或功能标识(一个类似于名称空间的 URI 和值)。功能采用 Boolean 值，而特性采用“对象”。

XMLReaderFactory 为方便创建不同的 XMLReader 而提供，最常用的 XMLReaderFactory 功能如下。

(1) http://xml.org/sax/features/namespaces：所有 SAX 语法分析器都能识别它。如果将它设置为 true(默认值)，则在调用 ContentHandler 的方法时，语法分析器将识别出名称空间并解析前缀。

(2) http://xml.org/sax/features/validation：它是可选的。如果将它设置为 true，则验证语法分析器将验证该文档。非验证语法分析器忽略该功能。

2. XMLReaderFactory

XMLReaderFactory 创建语法分析器对象。它定义 createXMLReader()的两个版本：一个采用语法分析器的类名作为参数，另一个从 org.xml.sax.driver 系统特性中获得类名称。

对于 Xerces，类是 org.apache.xerces.parsers.SAXParser。应该使用 XMLReaderFactory，因为它易于切换至另一种 SAX 语法分析器。实际上，只需要更改一行，然后重新编译。

```
XMLReaderparser=XMLReaderFactory.createXMLReader(
"org.apache.xerces.parsers.SAXParser");
```

为获得更大的灵活性，应用程序可以从命令行读取类名或使用不带参数的 createXMLReader()。因此，甚至可以不重新编译就更改语法分析器。

3. InputSource

InputSource 控制语法分析器如何读取文件，包括 XML 文档和实体。在大多数情况下，文档是从 URL 装入的。但是，有特殊需求的应用程序可以覆盖 InputSource。例如，还可以用来从数据库中装入文档。

4. ContentHandler

ContentHandler 是最常用的 SAX 接口，因为它定义了 XML 文档的事件。ContentHandler 声明以下几个事件。

(1) startDocument()/endDocument()：通知应用程序文档的开始或结束。

(2) startElement()/endElement()：通知应用程序标记的开始或结束。属性作为 Attributes

参数传递(请参阅“5. 属性”小节)。即使只有一个标记，“空”元素(例如，<imghref="logo.gif"/>)也生成 startElement()和 endElement()。

(3) startPrefixMapping()/endPrefixMapping()：通知应用程序名称空间作用域。您几乎不需要该信息，因为当 http://xml.org/sax/features/namespaces 为 true 时，语法分析器已经解析了名称空间。

(4) characters()/ignorablewhitespace()：当语法分析器在元素中发现文本(已经过语法分析的字符数据)时，characters()/ignorableWhitespace()会通知应用程序。要知道，语法分析器负责将文本分配到几个事件(更好地管理其缓冲区)。ignorableWhitespace 事件用于由 XML 标准定义的可忽略空格。

(5) processingInstruction()：将处理指令通知应用程序。

(6) skippedEntity()：通知应用程序已经跳过了一个实体(即，当语法分析器未在 DTD/schema 中发现实体声明时)。

(7) setDocumentLocator()：将 Locator 对象传递到应用程序；请参阅后面的 Locator 一节。请注意，不需要 SAX 语法分析器提供 Locator，但是如果它提供了，则必须在任何其他事件之前激活该事件。

5. 属性

在 startElement()事件中，应用程序在 Attributes 参数中接收属性列表。

```
Stringattribute=attributes.getValue("","price");
```

Attributes 定义下列方法。

- getValue(i)/getValue(qName)/getValue(uri,localName)：返回第 i 个属性值或给定名称的属性值。
- getLength()：返回属性数目。
- getQName(i)/getLocalName(i)/getURI(i)：返回限定名(带前缀)、本地名(不带前缀)和第 i 个属性的名称空间 URI。
- getType(i)/getType(qName)/getType(uri,localName)：返回第 i 个属性的类型或者给定名称的属性类型。类型为字符串，即在 DTD 所使用的：CDATA、ID、IDREF、IDREFS、NMTOKEN、NMTOKENS、ENTITY、ENTITIES 或 NOTATION。

注意： Attributes 参数仅在 startElement()事件期间可用。如果在事件之间需要它，则用 AttributesImpl 复制一个。

6. 定位器

Locator 为应用程序提供行和列的位置。不需要语法分析器来提供 Locator 对象。Locator 定义下列方法。

- getColumnNumber()：返回当前事件结束时所在的那一列。在 endElement()事件中，它将返回结束标记所在的最后一列。
- getLineNumber()：返回当前事件结束时所在的行。在 endElement()事件中，它将返回结束标记所在的行。

- getPublicId()：返回当前文档事件的公共标识。
- getSystemId()：返回当前文档事件的系统标识。

7. DTDHandler

DTDHandler 声明两个与 DTD 语法分析器相关的事件。具体如下。

- notationDecl()：通知应用程序已经声明了一个标记。
- nparsedEntityDecl()：通知应用程序已经发现了一个未经过语法分析的实体声明。

8. EntityResolver

EntityResolver 接口仅定义一个事件 resolveEntity()，它返回 InputSource(在另一章将讨论)。因为 SAX 语法分析器已经可以解析大多数 URL，所以很少有应用程序实现 EntityResolver。例外情况是目录文件(在另一章中讨论)，它将公共标识解析成系统标识。如果在应用程序中需要目录文件，请下载 NormanWalsh 的目录软件包(请参阅参考资料)。

9. ErrorHandler

ErrorHandler 接口定义错误事件。处理这些事件的应用程序可以提供定制错误处理。安装了定制错误处理器后，语法分析器不再抛出异常。抛出异常是事件处理器的责任。ErrorHandler 接口定义了与错误的三个级别或严重性对应的方法。

- warning()：警示那些不是由 XML 规范定义的错误。例如，当没有 XML 声明时，某些语法分析器发出警告。它不是错误(因为声明是可选的)，但是它可能值得注意。
- error()：警示那些由 XML 规范定义的错误。
- fatalError()：警示那些由 XML 规范定义的致命错误。

10. SAXException

SAX 定义的大多数方法都可以抛出 SAXException。当对 XML 文档进行语法分析时，SAXException 通知一个错误。错误可以是语法分析错误也可以是事件处理器中的错误。要想报告来自事件处理器的其他异常，可以将异常封装在 SAXException 中。

9.3　开发一个 RSS 订阅程序

在使用 RSS 订阅时，常常通过网站提供的“订阅 RSS”连接或小图标实现，当单击连接后，会弹出包含 RSS 内容的页面，此页面的网址是网站的 RSS 网址。当连接到这个网址后，服务器端会返回 RSS 标准规格的 XML 文件，只要按照统一格式来解析 XML 文件，就可以得到 RSS 内的相关信息。本节将通过一个具体实例的实现过程，讲解在 Android 系统中开发一个 RSS 项目的基本过程。

实　例	功　能	源码路径
实例 9-1	开发一个 RSS 系统	下载路径:\daima\9\RSSC

在本实例中，用户只需要输入一个 RSS feed 网址，通过 SAX Parser 解析后就可以直接在手机上浏览在线实时新闻。本实例的具体实现流程如下。

9.3.1 实现界面布局文件

(1) 编写主布局文件 main.xml，上方显示文字“设置 RSS 连接”，中间显示一个可输入文本框，下方显示一个按钮。主要代码如下：

```
<EditText
  android:id="@+id/myEdit"
  android:layout_width="280px"
  android:layout height="wrap content"
  android:text="http://"
  android:textSize="12sp"
  android:layout x="20px"
  android:layout y="42px"
>
</EditText>
<TextView
  android:id="@+id/myText"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:text="@string/str title"
  android:textColor="@drawable/black"
  android:textSize="16sp"
  android:layout x="20px"
  android:layout y="12px"
>
</TextView>
<Button
  android:id="@+id/myButton"
  android:layout width="86px"
  android:layout height="46px"
  android:text="@string/str_button"
  android:textColor="@drawable/black"
  android:layout x="100px"
  android:layout y="112px"
>
</Button>
```

(2) 编写布局文件 newslist.xml，在里面设置了一个 ListView 控件，用于列表显示获取的 RSS 信息标题。主要代码如下：

```
<TextView
  android:id="@+id/myText"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:padding="5px"
  android:textSize="18sp"
  android:textColor="@drawable/blue"
```

```
>
</TextView>
<ListView
  android:id="@android:id/list"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
>
</ListView>
```

(3) 编写布局文件 news_row.xml，在其中设置一个 TextView 控件，用于显示某一条 RSS 信息。主要代码如下：

```
<ImageView android:id="@+id/icon"
  android:layout_width="20dip"
  android:layout_height="20dip"
  android:src="@drawable/news"
>
</ImageView>
<TextView android:id="@+id/text"
  android:layout_gravity="center_vertical"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textColor="@drawable/black"
>
</TextView>
```

(4) 编写布局文件 newscontent.xml，在其中设置三个 TextView 控件，用于显示某条 RSS 信息的详细内容。主要代码如下：

```
<TextView
  android:id="@+id/myTitle"
  android:layout_width="300px"
  android:layout_height="wrap_content"
  android:textSize="16sp"
  android:layout_x="10px"
  android:layout_y="12px"
  android:textColor="@drawable/blue"
>
</TextView>
<TextView
  android:id="@+id/myDesc"
  android:layout_width="300px"
  android:layout_height="120px"
  android:layout_x="10px"
  android:layout_y="70px"
  android:textColor="@drawable/black"
>
</TextView>
<TextView
  android:id="@+id/myLink"
  android:layout_width="300px"
  android:layout_height="wrap_content"
```

```
  android:layout x="10px"
  android:layout y="210px"
 >
 </TextView>
```

9.3.2 实现主程序文件

(1) 编写主程序文件 RSSC.java，功能是以 EditText 来作为输入 RSS 连接组件，当输入网址后，单击“解析”按钮后，按钮的 onClick 会被触发，运行 EditText 的空白检查。当检查无误后，将输入的网址写入 Bundle 对象中，再将 Bundle 对象 assign 给 Intent，并通过 startActivityForResult()来触发 RSSC_1 这个 Activity。文件 RSSC.java 的实现代码如下：

```
package irdc.RSSC;

/* import 相关 class */
import irdc.RSSC.R;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class RSSC extends Activity
{
  /* 变量声明 */
  private Button mButton;
  private EditText mEditText;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /* 初始化对象 */
    mEditText=(EditText) findViewById(R.id.myEdit);
    mButton=(Button) findViewById(R.id.myButton);
    /* 设置 Button 的 onClick 事件 */
    mButton.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        String path=mEditText.getText().toString();
        if(path.equals(""))
        {
          showDialog("网址不可为空白!");
```



```
      }
      else
      {
        /* new 一个 Intent 对象，并指定 class */
        Intent intent = new Intent();
        intent.setClass(RSSC.this,RSSC 1.class);

        /* new 一个 Bundle 对象，并将要传递的数据传入 */
        Bundle bundle = new Bundle();
        bundle.putString("path",path);
        /* 将 Bundle 对象 assign 给 Intent */
        intent.putExtras(bundle);
        /* 调用 Activity RSSC_1 */
        startActivityForResult(intent,0);
      }
    }
  });
}

/* 覆盖 onActivityResult()*/
@Override
protected void onActivityResult(int requestCode,int resultCode,
                                Intent data)
{
  switch (resultCode)
  {
    case 99:
      /* 返回错误时以 Dialog 显示 */
      Bundle bunde = data.getExtras();
      String error = bunde.getString("error");
      showDialog(error);
      break;
    default:
      break;
   }
 }

/* 显示 Dialog 的方法 */
private void showDialog(String mess){
  new AlertDialog.Builder(RSSC.this).setTitle("Message")
  .setMessage(mess)
  .setNegativeButton("确定", new DialogInterface.OnClickListener()
  {
    public void onClick(DialogInterface dialog, int which)
    {
    }
  })
  .show();
}
}
```

(2) 编写文件 RSSC_1.java，此文件是一个 ListActivity，是通过主程序 RSSC.java 来调用的，用于显示订阅的 RSS 内容列表。其实现流程如下。

① 引用相关 class，分别声明变量 TextView mText、title 和 li。具体代码如下：

```
package irdc.RSSC;

/* import 相关 class */
import irdc.RSSC.R;

import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ListView;
import android.widget.TextView;
import javax.xml.parsers.*;
import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;

public class RSSC_1 extends ListActivity
{
  /* 变量声明 */
  private TextView mText;
  private String title="";
  private List<News> li=new ArrayList<News>();
```

② 设置 layout 为 newslist.xml，取得 Intent 中的 Bundle 对象，并取得 Bundle 对象中的数据，然后调用 getRss()取得解析后的 List。具体代码如下：

```
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    /* 设置 layout 为 newslist.xml */
    setContentView(R.layout.newslist);

    mText=(TextView) findViewById(R.id.myText);
    /* 取得 Intent 中的 Bundle 对象 */
    Intent intent=this.getIntent();
    Bundle bunde = intent.getExtras();
    /* 取得 Bundle 对象中的数据 */
    String path = bunde.getString("path");
    /* 调用 getRss()取得解析后的 List */
    li=getRss(path);
    mText.setText(title);
    /* 设置自定义的 MyAdapter */
    setListAdapter(new MyAdapter(this,li));
  }
```



③ 定义 onListItemClick，定义监听 ListItem 被点击时要做的动作。先获取 News 对象，新建一个 Intent 对象，并指定其 class，然后新建一个 Bundle 对象，并将要传递的数据传入，接着将 Bundle 对象 assign 给 Intent，最后调用 Activity RSSC_2。具体代码如下：

```
/* 设置 ListItem 被点击时要做的动作 */
@Override
protected void onListItemClick(ListView l,View v,int position,
                              long id)
{
  /* 取得 News 对象 */
  News ns=(News)li.get(position);
  /* new 一个 Intent 对象，并指定 class */
  Intent intent = new Intent();
  intent.setClass(RSSC_1.this,RSSC_2.class);
  /* new 一个 Bundle 对象，并将要传递的数据传入 */
  Bundle bundle = new Bundle();
  bundle.putString("title",ns.getTitle());
  bundle.putString("desc",ns.getDesc());
  bundle.putString("link",ns.getLink());
  /* 将 Bundle 对象 assign 给 Intent */
  intent.putExtras(bundle);
  /* 调用 Activity RSSC_2 */
  startActivity(intent);
}
```

④ 定义方法 getRss(String path)来解析 XML。具体代码如下：

```
/* 解析 XML 的方法 */
private List<News> getRss(String path)
{
  List<News> data=new ArrayList<News>();
  URL url = null;
  try
  {
    url = new URL(path);
    /* 产生 SAXParser 对象 */
    SAXParserFactory spf = SAXParserFactory.newInstance();
    SAXParser sp = spf.newSAXParser();
    /* 产生 XMLReader 对象 */
    XMLReader xr = sp.getXMLReader();
    /* 设置自定义的 MyHandler 给 XMLReader */
    MyHandler myExampleHandler = new MyHandler();
    xr.setContentHandler(myExampleHandler);
    /* 解析 XML */
    xr.parse(new InputSource(url.openStream()));
    /* 取得 RSS 标题与内容列表 */
    data =myExampleHandler.getParsedData();
    title=myExampleHandler.getRssTitle();
  }
```

⑤ 如果有异常则输出错误提示对话框，具体代码如下：

```
    catch (Exception e)
    {
      /* 发生错误时返回 result 到上一个 activity */
      Intent intent=new Intent();
      Bundle bundle = new Bundle();
      bundle.putString("error",""+e);
      intent.putExtras(bundle);
      /* 错误的返回值设置为 99 */
      RSSC_1.this.setResult(99, intent);
      RSSC_1.this.finish();
    }
    return data;
  }
}
```

(3) 编写文件 RSSC_2.java，此 Activity 由 RSSC_1 唤起，用于显示上一个 Activity 所点击的新闻内容。当程序被唤起后，首先会从 Bundle 对象中获取 News 的 title、link 和 desc，并显示在画面中。并以 Linkify.addLinks()将 link 设置为一个 WEB_URLS 形式的链接。当用户点击链接后，会通过设置的网址直接打开 Web 浏览器来浏览网页。其具体实现代码如下：

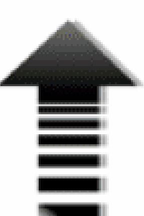
```
package irdc.RSSC;

/* import 相关 class */
import irdc.RSSC.R;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.text.util.Linkify;
import android.widget.TextView;

public class RSSC_2 extends Activity
{
  /* 变量声明 */
  private TextView mTitle;
  private TextView mDesc;
  private TextView mLink;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 设置 layout 为 newscontent.xml */
    setContentView(R.layout.newscontent);
    /* 初始化对象 */
    mTitle=(TextView) findViewById(R.id.myTitle);
    mDesc=(TextView) findViewById(R.id.myDesc);
    mLink=(TextView) findViewById(R.id.myLink);

    /* 取得 Intent 中的 Bundle 对象 */
    Intent intent=this.getIntent();
```



```
    Bundle bunde = intent.getExtras();
    /* 取得 Bundle 对象中的数据 */
    mTitle.setText(bunde.getString("title"));
    mDesc.setText(bunde.getString("desc")+"....");
    mLink.setText(bunde.getString("link"));
    /* 设置 mLink 为网页连接 */
    Linkify.addLinks(mLink,Linkify.WEB URLS);
  }
}
```

(4) 编写文件 News.java，在此定义了一个 JavaBean 类，用于存放每一篇新闻信息。每一个 News 对象代表了一条新闻，在 News 对象中定义了新闻的标题、描述、网站链接和发布时间 4 个属性。JavaBean 类中的方法都是以 setAAA()和 getAAA()方式来命名的，所以用 setAAA()来设置属性值，或通过 getAAA()来获取属性值。具体代码如下：

```
package irdc.RSSC;

public class News
{
  /* 新建变量初始值为空 */
  private String  title="";
  private String  link="";
  private String  desc="";
  private String  date="";

  public String getTitle()
  {
    return  title;
  }
  public String getLink()
  {
    return  link;
  }
  public String getDesc()
  {
    return  desc;
  }
  public String getDate()
  {
    return  date;
  }
  public void setTitle(String title)
  {
     title=title;
  }
  public void setLink(String link)
  {
     link=link;
  }
  public void setDesc(String desc)
  {
```

```
    desc=desc;
  }
  public void setDate(String date)
  {
    _date=date;
  }
}
```

(5) 编写文件 MyAdapter.java，在此定义了 Adapter 对象，它继承自 android.widget.BaseAdapter，用于设置 ListView 中要显示的信息，以 news_row.xml 作为 Layout。具体代码如下：

```
package irdc.RSSC;

/* import 相关 class */
import irdc.RSSC.R;

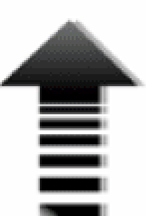
import java.util.List;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

/* 自定义的 Adapter，继承 android.widget.BaseAdapter */
public class MyAdapter extends BaseAdapter
{
  /* 变量声明  */
  private LayoutInflater mInflater;
  private List<News> items;

  /* MyAdapter 的构造器，传递两个参数  */
  public MyAdapter(Context context,List<News> it)
  {
    /* 参数初始化 */
    mInflater = LayoutInflater.from(context);
    items = it;
  }

  /* 因继承 BaseAdapter，需重写以下方法 */
  @Override
  public int getCount()
  {
    return items.size();
  }

  @Override
  public Object getItem(int position)
  {
    return items.get(position);
```



```
  }

  @Override
  public long getItemId(int position)
  {
    return position;
  }

  @Override
  public View getView(int position,View convertView,ViewGroup par)
  {
    ViewHolder holder;

    if(convertView == null)
    {
      /* 使用自定义的 news_row 作为 Layout */
      convertView = mInflater.inflate(R.layout.news row, null);
      /* 初始化 holder 的 text 与 icon */
      holder = new ViewHolder();
      holder.text = (TextView) convertView.findViewById(R.id.text);
      convertView.setTag(holder);
    }
    else
    {
      holder = (ViewHolder) convertView.getTag();
    }
    News tmpN=(News)items.get(position);
    holder.text.setText(tmpN.getTitle());

    return convertView;
  }

  /* class ViewHolder */
  private class ViewHolder
  {
    TextView text;
  }
}
```

(6) 编写文件 MyHandler.java，在此定义了 MyHandler 对象，它继承自 org.xml.sax.helpers.DefaultHandler，用于解析 XML 文件，并获取对应的信息。下面开始讲解文件 MyHandler.java 的具体实现流程。

① 引入相关 class，然后分别声明各个变量。具体代码如下：

```
package irdc.RSSC;

/* import 相关 class */
import java.util.ArrayList;
import java.util.List;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
```

```
import org.xml.sax.helpers.DefaultHandler;

public class MyHandler extends DefaultHandler
{
  /* 变量声明 */
  private boolean in item = false;
  private boolean in title = false;
  private boolean in link = false;
  private boolean in desc = false;
  private boolean in date = false;
  private boolean in mainTitle = false;
  private List<News> li;
  private News news;
  private String title="";
  private StringBuffer buf=new StringBuffer();
```

② 分别将转换成 List<News>的 XML 数据和解析出的 RSS title 返回，然后调用 startDocument()，开始解析操作。当解析结束时，调用 endDocument()。当解析到 Element 开头时，调用 startElement 方法。具体代码如下：

```
  /* 将转换成 List<News>的 XML 数据返回 */
  public List<News> getParsedData()
  {
    return li;
  }
  /* 将解析出的 RSS title 返回 */
  public String getRssTitle()
  {
    return title;
  }
  /* XML 文件开始解析时调用此方法 */
  @Override
  public void startDocument() throws SAXException
  {
    li = new ArrayList<News>();
  }
  /* XML 文件结束解析时调用此方法 */
  @Override
  public void endDocument() throws SAXException
  {
  }
  /* 解析到 Element 的开头时调用此方法 */
  @Override
  public void startElement(String namespaceURI, String localName,
              String qName, Attributes atts) throws SAXException
  {
    if (localName.equals("item"))
    {
      this.in item = true;
      /* 解析到 item 的开头时 new 一个 News 对象 */
      news=new News();
```



```
    }
    else if (localName.equals("title"))
    {
      if(this.in item)
      {
        this.in title = true;
      }
      else
      {
        this.in mainTitle = true;
      }
    }
    else if (localName.equals("link"))
    {
      if(this.in item)
      {
        this.in link = true;
      }
    }
    else if (localName.equals("description"))
    {
      if(this.in item)
      {
        this.in desc = true;
      }
    }
    else if (localName.equals("pubDate"))
    {
      if(this.in item)
      {
        this.in date = true;
      }
    }
  }
```

③ 当解析到 Element 的结尾时调用 endElement 方法，具体代码如下：

```
  /* 解析到 Element 的结尾时调用此方法 */
  @Override
  public void endElement(String namespaceURI, String localName,
                         String qName) throws SAXException
  {
    if (localName.equals("item"))
    {
      this.in_item = false;
      /* 解析到 item 的结尾时将 News 对象写入 List 中 */
      li.add(news);
    }
    else if (localName.equals("title"))
    {
      if(this.in item)
      {
```

```
      /* 设置 News 对象的 title */
      news.setTitle(buf.toString().trim());
      buf.setLength(0);
      this.in title = false;
    }
    else
    {
      /* 设置 RSS 的 title */
      title=buf.toString().trim();
      buf.setLength(0);
      this.in mainTitle = false;
    }
  }
  else if (localName.equals("link"))
  {
    if(this.in item)
    {
      /* 设置 News 对象的 link */
      news.setLink(buf.toString().trim());
      buf.setLength(0);
      this.in link = false;
    }
  }
  else if (localName.equals("description"))
  {
    if(in item)
    {
      /* 设置 News 对象的 description */
      news.setDesc(buf.toString().trim());
      buf.setLength(0);
      this.in desc = false;
    }
  }
  else if (localName.equals("pubDate"))
  {
    if(in item)
    {
      /* 设置 News 对象的 pubDate */
      news.setDate(buf.toString().trim());
      buf.setLength(0);
      this.in date = false;
    }
  }
}
```

④ 定义方法 characters 来获取 Element 开头和结尾中间的字符串，具体代码如下：

```
/* 取得 Element 的开头和结尾中间的字符串 */
@Override
public void characters(char ch[], int start, int length)
{
  if(this.in_item||this.in_mainTitle)
```

```
    {
      /* 将 char[]添加 StringBuffer */
      buf.append(ch,start,length);
    }
  }
}
```

执行后的效果如图 9-3 所示。在文本框中输入 RSS 网址 http://rss.sina.com.cn/news/marquee/ddt.xml，然后单击“解析”按钮，会在屏幕中列表显示 RSS 新闻，如图 9-4 所示。单击某条新闻后，会显示此新闻的简介，如图 9-5 所示。单击简介下面的链接后会显示此 RSS 的详细信息，如图 9-6 所示。



图 9-3　执行效果



图 9-4　RSS列表



图 9-5　某条RSS



图 9-6　RSS详情



9.4 开发一个 RSS 阅读器

从本节内容开始，将着重介绍本项目的具体实现过程。详细讲解各个代码的具体实现过程，并讲解其中的技巧和要点，使读者的水平更上一层楼。

实　例	功　能	源码路径
实例 9-2	开发一个 RSS 阅读器	下载路径:\daima\9\RSSREAD

9.4.1 建立实体类

一个 RSS 文件可以被认为是由一个 RSS 的一些描述性信息和里面的 item 元素组成的，例如下面关于 RSS 的描述性信息。

- title：标题信息。
- link：链接信息。
- description：描述信息。

item 中的信息如下。

- title：标题信息。
- link：链接信息。
- description：描述信息。
- pubDate：发布的日期。

在本项目实例中需要建立以下两个实体类。

- RSSFeed：用于和一个 RSS 的完整 XML 文件相对应。
- RSSItem：用于和一个 RSS 中的 Item 标签相对应。

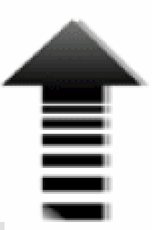
在解析 RSS 文件时，可以将文件里的信息解析出来放到实体类中，这样就可以直接操作该实体类了。下面开始讲解上述两个实体类的具体实现过程。

1. RSSFeed类

RSSFeed 类的功能是，建立和一个完整 XML 文件的对应，其中方法 addItem 用于将一个 RSSItem 添加到 RSSFeed 类中；方法 getAllItemsForListView 负责从 RSSFeed 类中生成 ListView 列表所需要的数据。RSSFeed 类的具体实现代码如下：

```
package com.rss reader.data;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Vector;

public class RSSFeed
{
```



```
    private String title = null;
    private String pubdate = null;
    private int itemcount = 0;
    private List<RSSItem> itemlist;


     public RSSFeed()
    {
        itemlist = new Vector(0);
    }
    public int addItem(RSSItem item)
    {
        itemlist.add(item);
        itemcount++;
        return itemcount;
    }
    public RSSItem getItem(int location)
    {
        return itemlist.get(location);
    }
    public List getAllItems()
    {
        return itemlist;
    }
    public List getAllItemsForListView(){
        List<Map<String, Object>> data = new ArrayList<Map<String, Object>>();
        int size = itemlist.size();
        for(int i=0;i<size;i++){
            HashMap<String, Object> item = new HashMap<String, Object>();
            item.put(RSSItem.TITLE, itemlist.get(i).getTitle());
            item.put(RSSItem.PUBDATE, itemlist.get(i).getPubDate());
            data.add(item);
        }
        return data;
    }
    int getItemCount()
    {
        return itemcount;
    }
    public void setTitle(String title)
    {
        this.title = title;
    }
    public void setPubDate(String pubdate)
    {
        this.pubdate = pubdate;
    }
    public String getTitle()
    {
        return title;
    }
```

```
    public String getPubDate()
    {
        return pubdate;
    }



}
```

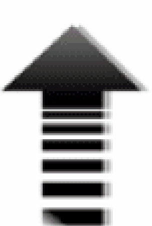
2. RSSItem类

RSSItem 类的功能是，用于和一个 RSS 中的 Item 标签相对应，其中的属性和 item 中的属性一样。RSSItem 类的具体实现代码如下：

```
package com.rss reader.data;

public class RSSItem
{
    public static final String TITLE="title";
    public static final String PUBDATE="pubdate";
    private String title = null;
    private String description = null;
    private String link = null;
    private String category = null;
    private String pubdate = null;


    public RSSItem()
    {
    }
    public void setTitle(String title)
    {
        this.title = title;
    }
    public  void setDescription(String description)
    {
        this.description = description;
    }
    public  void setLink(String link)
    {
        this.link = link;
    }
    public  void setCategory(String category)
    {
        this.category = category;
    }
    public  void setPubDate(String pubdate)
    {
        this.pubdate = pubdate;
    }
    public String getTitle()
```



```
    {
        return title;
    }
    public  String getDescription()
    {
        return description;
    }
    public  String getLink()
    {
        return link;
    }
    public String getCategory()
    {
        return category;
    }
    public String getPubDate()
    {
        return pubdate;
    }
    public String toString()
    {
        if (title.length() > 20)
        {
            return title.substring(0, 42) + "...";
        }
        return title;
    }
}
```

9.4.2　主程序文件ActivityMain.java

主程序文件 ActivityMain.java 是本项目的入口，在此 Activity 中得到了服务器端的 RSSFeed，经过解析后将其中的内容以 ListView 的形式列表显示。下面讲解其具体实现流程。

(1) 先引入相关 class 类，然后设置目标 RSS 的源地址为 http://feed.feedsky.com/woshiyigebing12345，最后通过 showListView()方法将获取的 RSS 信息以列表形式显示出来。具体代码如下：

```
package com.rss reader;

import java.net.URL;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;

import android.app.Activity;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;

import com.rss reader.data.RSSFeed;
import com.rss reader.data.RSSItem;
import com.rss reader.sax.RSSHandler;

public class ActivityMain extends Activity implements
OnItemClickListener
{
//  public final String RSS URL = "http://rubyjin.cn/blog/rss";

        public final String RSS URL =
        " http://feed.feedsky.com/woshiyigebing12345";

    public final String tag = "RSSReader";
    private RSSFeed feed = null;

    /** Called when the activity is first created. */

   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      setContentView(R.layout.main);
      feed = getFeed(RSS URL);
      showListView();

    }
```

(2) 定义方法 getFeed(String urlString)，用于得到一个 RSSFeed，即从服务器端请求了 RSS feed，并进行了解析，将解析后的内容都放在 RSSFeed 的一个实例中。上述解析过程是通过 SAX 实现的，具体流程如下。

① 新建工厂类 SAXParserFactory。

② 工厂类产生一个 SAX 解析类 SAXParser。

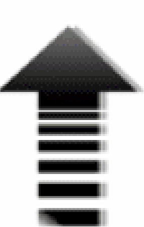
③ 从 SAXParser 中得到一个 XMLReader 实例，XMLReader 是一个接口，此接口中定义了一些解析 XML 的回调函数。

④ 把编写的 Handler 注册到 XMLReader 中去。

⑤ 将一个 XML 文档或资源变成一个 Java 可以处理的 InputStream 流后，解析工作开始。

方法 getFeed(String urlString)的具体代码如下：

```
   private RSSFeed getFeed(String urlString)
   {
    try
```



```
{
  URL url = new URL(urlString);
    /*新建工厂类 SAXParserFactory*/
     SAXParserFactory factory = SAXParserFactory.newInstance();
    /*工厂类产生一个 SAX 解析类 SAXParser */
     SAXParser parser = factory.newSAXParser();
    /*从 SAXParser 中得到一个 XMLReader 实例*/
     XMLReader xmlreader = parser.getXMLReader();
     /*把编写的 Handler 注册到 XMLReader 中 */
     RSSHandler rssHandler = new RSSHandler();
     xmlreader.setContentHandler(rssHandler);

     /*将一个 XML 文档或资源变成一个 Java 可以处理的 InputStream 流后，解析工作开始。*/
     InputSource is = new InputSource(url.openStream());

     xmlreader.parse(is);

     return rssHandler.getFeed();
}
catch (Exception ee)
{

    return null;
}
}
```

(3) 定义方法 showListView()来列表显示获取的 RSS，这样 ListView 和一个 SimpleAdapter 实现了绑定。具体代码如下：

```
private void showListView()
{
    ListView itemlist = (ListView) findViewById(R.id.itemlist);
    if (feed == null)
    {
     setTitle("访问的 RSS 无效");
     return;
    }
    SimpleAdapter adapter = new SimpleAdapter(this,
          feed.getAllItemsForListView(),
         android.R.layout.simple list item 2, new String[]
         { RSSItem.TITLE,RSSItem.PUBDATE },
          new int[] { android.R.id.text1 , android.R.id.text2});
    itemlist.setAdapter(adapter);
    itemlist.setOnItemClickListener(this);
    itemlist.setSelection(0);

}
```

(4) 定义方法 onItemClick，用于处理列表的单击事件，当单击后会显示此 RSS 信息的链接地址，用户单击后可以通过浏览器来到目标地址。具体代码如下：

```
    public void onItemClick(AdapterView parent, View v, int position, long id)
    {
     Intent itemintent = new Intent(this,ActivityShowDescription.class);

     Bundle b = new Bundle();
     b.putString("title", feed.getItem(position).getTitle());
     b.putString("description", feed.getItem(position).getDescription());
     b.putString("link", feed.getItem(position).getLink());
     b.putString("pubdate", feed.getItem(position).getPubDate());

     itemintent.putExtra("android.intent.extra.rssItem", b);
        startActivityForResult(itemintent, 0);
    }
```

9.4.3 实现ContentHandler

ContentHandler 是一个特殊的 SAX 接口，位于 org.xml.sax.ContentHandler。在我们解析 XML 时，大多数步骤都是固定不变的，但是关于 ContentHandler 的实现却是不同的。实现 ContentHandler 是解析 XML 中最重要、最关键的步骤之一。下面将讲解其具体实现流程。

(1) 声明 RSSHandler 类，声明继承与 DefaultHandler 的类。DefaultHandler 类是一个基类，此类里面最简单地实现了一个 ContentHandler，只需在其中重写里面的重要方法即可。具体代码如下：

```
package com.rss reader.sax;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

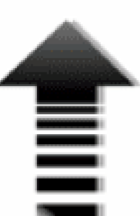
import android.util.Log;

import com.rss reader.data.RSSFeed;
import com.rss reader.data.RSSItem;

public class RSSHandler extends DefaultHandler
{

    RSSFeed rssFeed;
    RSSItem rssItem;
    String lastElementName = "";
    final int RSS_TITLE = 1;
    final int RSS LINK = 2;
    final int RSS DESCRIPTION = 3;
    final int RSS CATEGORY = 4;
    final int RSS PUBDATE = 5;

    int currentstate = 0;
```



```
public RSSHandler()
{
}

public RSSFeed getFeed()
{
    return rssFeed;
}
```

(2) 分别重写 startDocument()和 endDocument()。通常将正式解析前的初始化工作放到 startDocument()中，将一些收尾性工作放到 endDocument()中。具体代码如下：

```
public void startDocument() throws SAXException
{
    rssFeed = new RSSFeed();
    rssItem = new RSSItem();

}
public void endDocument() throws SAXException
{
}
```

(3) 重写 startElement，当 XML 解析器遇到 XML 文档流里面的 tag 时，将会调用此函数。在此函数内部通常是通过参数 localName 进行判断并进行一些操作处理的。具体代码如下：

```
public void startElement(String namespaceURI, String
   localName,String qName, Attributes atts) throws SAXException
{
    if (localName.equals("channel"))
    {
        currentstate = 0;
        return;
    }
    if (localName.equals("item"))
    {
        rssItem = new RSSItem();
        return;
    }
    if (localName.equals("title"))
    {
        currentstate = RSS TITLE;
        return;
    }
    if (localName.equals("description"))
    {
        currentstate = RSS DESCRIPTION;
        return;
    }
    if (localName.equals("link"))
```

```
        {
            currentstate = RSS_LINK;
            return;
        }
        if (localName.equals("category"))
        {
            currentstate = RSS_CATEGORY;
            return;
        }
        if (localName.equals("pubDate"))
        {
            currentstate = RSS_PUBDATE;
            return;
        }

        currentstate = 0;
    }
```

(4) 重写 endElement，此方法和 startElement 方法相对应，当解析 tag 完毕后执行此方法。如果解析一个 item 节点结束，就将 RSSItem 添加到 RSSFeed 中去。具体代码如下：

```
    public void endElement(String namespaceURI, String localName, String
qName) throws SAXException
    {

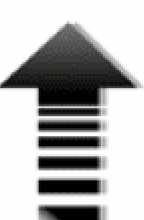
        //如果解析一个 item 节点结束，就将 RSSItem 添加到 RSSFeed 中。
        if (localName.equals("item"))
        {
            rssFeed.addItem(rssItem);
            return;

        }
    }
```

(5) 重写 characters，此方法是一个回调方法，当解析完 startElement 方法后，解析完节点内容后会执行此方法，并且参数 ch[]就是节点的内容。具体代码如下：

```
    public void characters(char ch[], int start, int length)
    {
        String theString = new String(ch,start,length);

        switch (currentstate)
        {
            case RSS_TITLE:
                rssItem.setTitle(theString);
                currentstate = 0;
                break;
            case RSS_LINK:
                rssItem.setLink(theString);
                currentstate = 0;
                break;
            case RSS_DESCRIPTION:
```



```
                rssItem.setDescription(theString);
                currentstate = 0;
                break;
            case RSS CATEGORY:
                rssItem.setCategory(theString);
                currentstate = 0;
                break;
            case RSS PUBDATE:
                rssItem.setPubDate(theString);
                currentstate = 0;
                break;
            default:
                return;
        }

    }
}
```

9.4.4　主程序文件ActivityShowDescription.java

主程序文件 ActivityShowDescription.java 的功能是，显示某列表信息的详细信息。当单击列表中的某一项后，会进入到此界面。如果程序出错，则 content 显示出错提示；运行正确则在 content 中分别显示 title、pubdate 和 description。具体代码如下：

```
package com.rss reader;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.content.Intent;
import android.view.*;

public class ActivityShowDescription extends Activity {
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.showdescription);
        String content = null;
        Intent startingIntent = getIntent();

        if (startingIntent != null) {
            Bundle bundle = startingIntent
                    .getBundleExtra("android.intent.extra.rssItem");
            if (bundle == null) {
                content = "不好意思程序出错啦";
            } else {
                content = bundle.getString("title") + "\n\n"
                        + bundle.getString("pubdate") + "\n\n"
                        + bundle.getString("description").replace('\n', ' ')
                        + "\n\n 详细信息请访问以下网址:\n" +
```

```
                    bundle.getString("link");
            }
        } else {
            content = "不好意思程序出错啦";

        }

        TextView textView = (TextView) findViewById(R.id.content);
        textView.setText(content);

        Button backbutton = (Button) findViewById(R.id.back);

        backbutton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }
}
```

9.4.5　主布局文件main.xml

主布局文件 main.xml 用于定义系统初始主界面，即列表显示获取的 RSS 信息。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
<ListView
    android:layout width="fill parent"
    android:layout height="fill parent"
    android:id="@+id/itemlist"

    />
</LinearLayout>
```

9.4.6　详情布局文件showdescription.xml

当用户单击列表信息后，会进入信息详情界面，此界面是由布局文件 showdescription.xml 定义的。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
```



```
<TextView
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:autoLink="all"
    android:text=""
    android:id="@+id/content"
    android:layout weight="1.0"
    />
<Button
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:text="返回"
     android:id="@+id/back"
     />

</LinearLayout>
```

至此，整个实例介绍完毕。运行后将获取指定 RSS 中的信息，如图 9-7 所示。单击某条信息后会显示此信息的相关描述性信息，如图 9-8 所示。



图 9-7　执行效果



图 9-8　相关描述性信息

单击图 9-8 中间的链接后，能够显示此条 RSS 的详细信息，如图 9-9 所示。

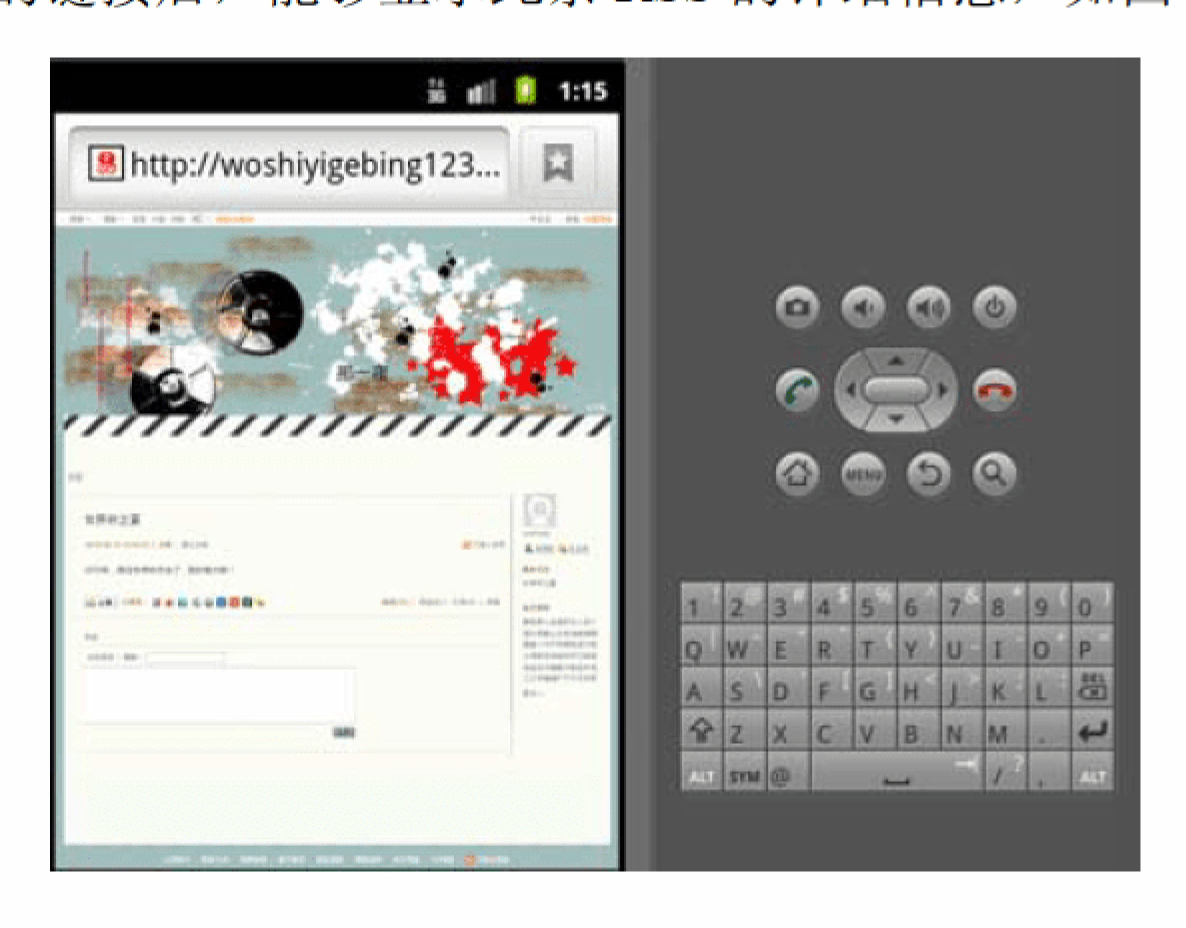


图 9-9　详细信息

本实例默认显示的是博客 http://woshiyigebing12345.blog.163.com/中的信息。读者也可以指定显示其他 RSS 信息，在使用时可以登录 http://www.feedsky.com/来设置不同的 RSS 订阅。具体设置流程如下。

(1) 登录 http://www.feedsky.com/主界面，如图 9-10 所示。



图 9-10 feedsky.com主界面

(2) 在图 9-10 顶部的文本框中输入要显示信息的博客地址、Feed 地址或 QQ 号码，然后单击“下一步”按钮，如图 9-11 所示。



图 9-11 输入设置的博客、QQ或Feed地址

(3) 在弹出的如图 9-12 所示的界面中分别输入“名称”、“描述”和 tag，并设定永久性 Feed 地址。



图 9-12 设置Feed界面

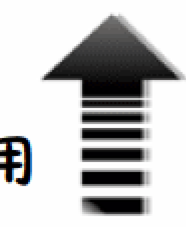


图 9-12 中的永久 Feed 地址就是 RSS 的源地址，这样就可以将此地址添加到实例中，从而显示此地址的 RSS 资源信息，也就是显示博客 http://woshiyigebing12345.blog.163.com/ 中的信息。

Android

第 10 章

在Android中开发电子邮件应用

自从互联网诞生以来，电子邮件就成为吸引用户应用网络的主要原因之一。无论是亲朋好友之间的祝福和交流，还是商务中的信息往来，都离不开电子邮件。自从进入信息时代之后，E-mail 就成为网络中的弄潮儿，深受人们的青睐。本章将详细介绍在 Android 平台中开发电子邮件应用的基本知识，为读者步入本书后面知识的学习打下基础。

10.1　使用 Android 的内置邮件系统

其实无须开发人员伤脑筋，在 Android 系统中 Google 内置了功能强大的 Gmail 邮件系统。我们只需对其进行相关设置，就可以使用内置的邮件系统收发邮件。

10.1.1　Android邮件客户端配置

在 Android 操作系统中除了有 Gmail 外，还包含了一个功能强大的 E-mail 电子邮件客户端，支持 POP3 和 IMAP 协议。在设置 Android 邮件客户端之前需要确保手机已连通无线或有线网络。

(1) 进入 Android 操作系统的应用程序界面，找到 E-mail 图标，如图 10-1 所示。

(2) 首次进入需要设置一个电子邮箱账户及密码，输入我们的免费邮箱账号和密码，例如 21CN 的邮箱，然后单击“下一步”按钮，如图 10-2 所示。



图 10-1　找到E-mail图标



图 10-2　输入邮箱信息

(3) 在“添加新电子邮件帐户”界面中选择账户类型，21CN 免费邮箱支持 POP3 及 IMAP 服务器接收，在此根据需要选择 POP3 或是 IMAP 方式接收邮件，如图 10-3 所示。

(4) 如果账户类型选择的是 POP3，则在此输入用户名(完整的邮件地址)、密码、21CN 免费邮箱的 POP3 服务器名称“pop.21cn.com”及端口“110”，如图 10-4 所示。

(5) 因为 21CN 免费邮箱支持 POP 的 SSL 加密连接方式，若需使用 SSL 加密连接，单击“安全类型”的下拉按钮，选择 SSL，设置 POP 方式的接收邮件服务器“pop.21cn.com”，安全类型为 SSL，端口号为 995，如图 10-5 所示。

(6) 如果账户类型选择的是 IMAP，则输入用户名(完整的邮件地址)、密码、21CN 免费邮箱的 IMAP 服务器名称“imap.21cn.com”及端口“143”，如图 10-6 所示。





图 10-3　账户类型



图 10-4　接收服务器设置



图 10-5　设置SSL加密连接方式

(7) 同样因为 21CN 免费邮箱支持 IMAP 的 SSL 加密连接方式，若需使用 SSL 加密连接，单击“安全类型”下拉按钮，选择 SSL，设置 IMAP 方式的接收邮件服务器“imap.21cn.com”，安全类型为 SSL，端口号为 993，如图 10-7 所示。

(8) 配置好接收服务器后单击“下一步”按钮，此时手机将与服务器相连以检查接收服务器的设置，如图 10-8 所示。

(9) 如果检查接收服务器的设置正确，则进入“外发服务器设置”界面，在此输入 21CN 免费邮箱的 SMTP 服务器“smtp.21cn.com”、端口“25”、用户名(完整的邮件地址)和密码，同时必须开启“需要登录”，设置完成后单击“下一步”按钮，如图 10-9 所示。



图 10-6　接收服务器设置



图 10-7　设置SSL加密连接



图 10-8　检查设置



图 10-9　外发服务器设置

(10) 进入“帐户选项”设置界面，单击“收件箱检查频率”下拉按钮可以进行收件箱频率的设置，选择“一律不”选项对于 CMNET 的 GPRS 用户比较合适，如果使用 Wi-Fi 可以选择“每隔 5 分钟”、“每隔 10 分钟”等选项，最后单击“下一步”按钮，如图 10-10 所示。

(11) 设置邮箱的昵称，即发件人姓名等属性，比如输入“Kelly”，最后单击“完成”按钮完成设置，如图 10-11 所示。

此时就会在“收件箱”中看到电子邮件了。

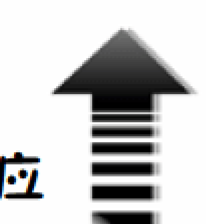






图 10-10　“收件箱检查频率设置”界面



图 10-11　设置电子邮件

10.1.2　调用内置邮件系统在发送短信时实现E-mail通知

在 Android 系统中，可以用编程的方式调用内置邮件系统来发送邮件。在具体实现时，需要用 Intent 来配合实现。为了说明具体原理，我们先看下面的一段代码。

```
Intent intent = new Intent(android.content.Intent.ACTION SEND);
intent.putExtra(android.content.Intent.EXTRA EMAIL, new
String[]{"test@test.com"});
intent.putExtra(android.content.Intent.EXTRA SUBJECT, "SUBJECT");
intent.putExtra(android.content.Intent.EXTRA TEXT, "TEXT");
intent.setType("text/html");
```

```
startActivity(Intent.createChooser(intent, "Chooser"));
Intent intent = new Intent(android.content.Intent.ACTION_SEND);
```

在上述代码中，将所有含有发送功能的 APP 做成一个列表以供选择，然后用 putExtra()方法将邮件的各个部分发送 E-mail 程序。方法 putExtra()的语法格式如下：

```
intent.putExtra(android.content.Intent.EXTRA_STREAM,URL url);
```

第一个参数 EXTRA_STREAM 表示传输的数据，具体说明如下。

- EXTRA_EMAIL：发送的是收件人地址。
- EXTRA_SUBJECT：邮件标题。
- EXTRA_TEXT：邮件文本内容。
- EXTRA_STREAM：邮件附件。

第二个参数 url 表示传递对象的 URL 地址。

下面将通过一个具体实例来讲解在发送短信时实现 E-mail 邮件通知的过程。

实　例	功　能	源码路径
实例 10-1	在收到短信时实现 Email 邮件通知	下载路径:\daima\10\tong

在本实例中，当用户收到一条短信后，先用 Toast 提示获取了短信，然后使用 E-mail 发送提示到用户的邮箱中，这样就可以将重要的短信放在邮箱中保存，从而不用担心短信容量的问题了。

在具体实现上，先在后台设计一个 BroadcastReseiver 用于等待接收短信。当接收到短信后，使用 Bundle 方式封装短信内容，然后通过 Intent 方式返回给主程序 Activity。因为 Receiver 无法直接发送 E-mail，所以需要将控制权返回给主程序，通过主程序来运行发送 E-mail 的工作。当主程序收到 Bundle 后，会以 Bundle.getString 的方法来取得返回短信的内容，然后以 Intent.setType("plain/text")来设置要打开的 Intent 类型，并以关键程序 Intent.putExtra(android.content.Intent.EXTRA_EMAIL,strEmailReciver)来指定要打开的是 E-mail 所需要的 Extra 参数，当 Android 系统收到这些参数后，就会打开内置的 E-mail 发送程序。读者在此需要注意的是，在模拟器运行后会显示“No application can perform this action”的提示，而在真实机器上不会出现此问题。

本实例的具体实现流程如下。

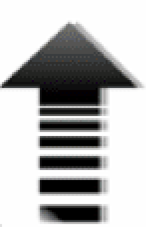
(1) 编写文件 tong.java，其具体实现流程如下。

① 分别声明一个 TextView、String 数组和两个文本字符串变量，主要代码如下：

```
/*声明一个 TextView、String 数组与两个文本字符串变量*/
private TextView mTextView1;
public String[] strEmailReciver;
public String strEmailSubject;
public String strEmailBody;
```

② 通过 findViewById 构造器来创建 TextView 对象，并通过 TextView 来显示“等待中...”的提示。主要代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
```



```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*通过 findViewById 构造器创建 TextView 对象*/
    mTextView1 = (TextView) findViewById(R.id.myTextView1);
    mTextView1.setText("等待中...");
```

③ 通过 try 语句获取短信传来的 bundle 堆信息，并取出 bunde 中的字符串，然后自定义 Intent 来寄送 E-mail 邮件信息，同时设置邮件格式为“plain/text”，并分别取得 EditText01、EditText02、EditText03 和 EditText04 的值作为收件人的地址、附件、主题和正文，最后将取得的字符串放入 mEmailIntent 中。主要代码如下：

```
    try {
      /*取得短信传来的 bundle 堆信息*/
      Bundle bunde = this.getIntent().getExtras();
      if (bunde!= null)
      {
        /*将 bunde 内的字符串取出*/
        String sb = bunde.getString("STR INPUT");
        /*自定义一个 Intent 来运行寄送 E-mail 的工作*/
        Intent mEmailIntent =
        new Intent(android.content.Intent.ACTION SEND);
        /*设置邮件格式为"plain/text"*/
        mEmailIntent.setType("plain/text");

        /*取得 EditText01、EditText02、EditText03、EditText04 的值作为收件人的
           地址、附件、主题、正文*/
        strEmailReciver =new String[]{"jay.mingchieh@gmail.com"};
        strEmailSubject = "你有一封短信!!";
        strEmailBody = sb.toString();

        /*将取得的字符串放入 mEmailIntent 中*/
        mEmailIntent.putExtra(android.content.Intent.EXTRA EMAIL,
           strEmailReciver);
        mEmailIntent.putExtra(android.content.Intent.EXTRA SUBJECT,
           strEmailSubject);
        mEmailIntent.putExtra(android.content.Intent.EXTRA TEXT,
           strEmailBody);
        startActivity(Intent.createChooser(mEmailIntent,
           getResources().getString(R.string.str message)));
      }
      else
      {
        finish();
      }
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

(2) 编写文件 SMSreceiver.java，其具体实现流程如下。

① 引用 BroadcastReceiver 类，使用 telephoney.gsm.SmsMessage 来接收短信，使用 Toast 类来通知用户收到短信，主要代码如下：

```
/*使用 telephoney.gsm.SmsMessage 来收取短信*/
import android.telephony.gsm.SmsMessage;
/*使用 Toast 类来告知用户收到短信*/
import android.widget.Toast;
```

② 自定义继承自 BroadcastReceiver 类，用于侦听系统服务广播的信息，然后声明静态字符串并作为 Action 启动短信的依据，主要代码如下：

```
public class SMSreceiver extends BroadcastReceiver
{
  * android.provider.Telephony.SMS RECEIVED
  private static final String mACTION =
  "android.provider.Telephony.SMS RECEIVED";

  private String str_receive="收到短信!";
```

③ 定义方法 onReceive(Context context, Intent intent)来获取短信，先通过 if 语句判断传来的 Intent 是否为短信，如果为短信则建构一字符串集合变量 sb 并接收由 Intent 传来的数据。主要代码如下：

```
  public void onReceive(Context context, Intent intent)
  {
    Toast.makeText(context, str receive.toString(),
    Toast.LENGTH_LONG).show();
    /*判断传来的 Intent 是否为短信*/
    if (intent.getAction().equals(mACTION))
    {
      /*建构一字符串集合变量 sb*/
      StringBuilder sb = new StringBuilder();
      /*接收由 Intent 传来的数据*/
      Bundle bundle = intent.getExtras();
```

④ 使用 if 语句判断在 Intent 中是否有数据，用 pdus 作为 Android 内置短信参数 identifier，并通过 bundle.get("")返回一包含 pdus 的对象。主要代码如下：

```
      /*判断 Intent 是有数据*/
      if (bundle != null)
      {
        Object[] myOBJpdus = (Object[]) bundle.get("pdus");
```

⑤ 构造短信对象 array，然后依据收到的对象长度来创建 array 的大小。主要代码如下：

```
        SmsMessage[] messages = new SmsMessage[myOBJpdus.length];

        for (int i = 0; i<myOBJpdus.length; i++)
        {
          messages[i] = SmsMessage.createFromPdu((byte[]) myOBJpdus[i]);
        }
```

⑥ 分别获取收信人的电话号码，并将传来的信息保存在 BODY。主要代码如下：

```
    for (SmsMessage currentMessage : messages)
    {
      sb.append("接收到来自:\n");
      /* 收信人的电话号码 */
      sb.append(currentMessage.getDisplayOriginatingAddress());
      sb.append("\n------传来的短信------\n");
      /* 取得传来信息的 BODY */
      sb.append(currentMessage.getDisplayMessageBody());
      Toast.makeText
      (
        context, sb.toString(), Toast.LENGTH_LONG
      ).show();
    }
  }
```

⑦ 使用 Notification(Toase)提醒来显示提示信息，主要代码如下：

```
Toast.makeText
(
  context, sb.toString(), Toast.LENGTH_LONG
).show();
```

⑧ 返回主 Activity，然后自定义一个 Bundle，将短信信息以 putString()方法存入自定义的 bundle 内，最后设置 Intent 的 Flag 以一个全新的 task 来运行。主要代码如下：

```
      Intent i = new Intent(context, GMail.class);
      /*自定义一个 Bundle*/
      Bundle mbundle = new Bundle();
      /*将短信信息以 putString()方法存入自定义的 bundle 内*/
      mbundle.putString("STR_INPUT",  sb.toString());
      /*将自定义 bundle 写入 Intent 中*/
      i.putExtras(mbundle);
      /*设置 Intent 的 Flag 以一个全新的 task 来运行*/
      i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
      context.startActivity(i);
    }
  }
}
```

(3) 编写文件 AndroidManifest.xml，向系统注册一个常驻的 BroadcastReseiver，并设置这个 Reseiver 的 intent-filter，让其 SMSreceiver 针对收到短信事件做出反应，并声明 android.permission.RECEIVE_SMS 权限。主要代码如下：

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
   </activity>
   <!-- 建立 receiver 来侦听系统广播信息 -->
   <receiver android:name="irdc.tong.SMSreceiver">
```

```
    <!-- 设置要捕捉的信息名是provider中的Telephony.SMS_RECEIVED -->
  <intent-filter>
    <action
      android:name="android.provider.Telephony.SMS_RECEIVED" />
  </intent-filter>
  </receiver>
    </application>
<uses-permission android:name="android.permission.RECEIVE_SMS">
</uses-permission>
</manifest>
```

实例执行后，如果收到一条短信则会显示提示信息，并自动生成一条邮件提示，如图 10-12 所示。

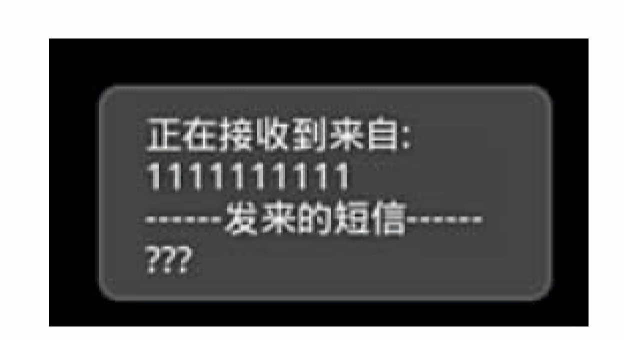


图 10-12　运行效果

10.1.3　调用内置邮件系统在来电时实现自动邮件通知

在实例 10-1 中，介绍了来短信后自动发送邮件通知的实现过程。同理也可以编写一个程序，在来电时实现自动邮件通知功能。

实　例	功　能	源码路径
实例 10-2	在来电时实现 E-mail 邮件通知	下载路径:\daima\10\dian

在本实例中，通过 TelephoneManage 来判断来电状态，并实现来电通知。程序通过 E-mail 来通知来电记录，本实例继承了前面的实例，并再次对 PhoneCallListener 来判断电话事件，并根据来电状态发送 E-mail。

本实例的具体实现流程如下。

(1) 编写文件 strings.xml，设置在屏幕中显示的文本，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello"></string>
  <string name="app_name"></string>
  <string name="str_button1">获取信息</string>
  <string name="str_CALL_STATE_IDLE">待机状态中</string>
  <string name="str_CALL_STATE_OFFHOOK">通话中...</string>
  <string name="str_CALL_STATE_RINGING">有电话...</string>
  <string name="str_EmailBody">有电话....</string>
  <string name="str_message">发信中....</string>
</resources>
```

(2) 编写文件 dian.java，具体实现流程如下。

① 定义 TelephonyManager 对象 telMgr，通过此对象获取 TELEPHONY_SERVICE 系



统信息，主要代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  mPhoneCallListener phoneListener=new mPhoneCallListener();
  /*对象 telMgr，用于获取 TELEPHONY_SERVICE 系统*/
  TelephonyManager telMgr = (TelephonyManager)getSystemService
              (TELEPHONY SERVICE);
  telMgr.listen(phoneListener, mPhoneCallListener.
              LISTEN CALL STATE);
  mTextView1 = (TextView)findViewById(R.id.myTextView1);
}
```

② 使用 PhoneCallListener 来侦听电话状态更改事件，方法 onCallStateChanged 的功能如下。

- 分别获取电话待机状态、通话状态和来电状态。
- 显示号码。
- 有电话时发送邮件。
- 设置收信人邮箱地址。
- 设置邮件标题。
- 设置邮件内容。
- 实现发信处理。

上述功能的主要代码如下：

```
public class mPhoneCallListener extends PhoneStateListener
{
  @Override
  public void onCallStateChanged(int state, String incomingNumber)
  {
    switch(state)
    {
      /*  获取电话待机状态*/
      case TelephonyManager.CALL STATE IDLE:
        mTextView1.setText(R.string.str CALL STATE IDLE);
        break;
      /*  获取电话通话状态*/
      case TelephonyManager.CALL STATE OFFHOOK:
        mTextView1.setText(R.string.str CALL STATE OFFHOOK);
        break;
      /*  获取电话来电状态*/
      case TelephonyManager.CALL STATE RINGING:
        mTextView1.setText
        (
          /*显示号码*/
          getResources().getText(R.string.str CALL STATE RINGING)+
          incomingNumber
```

```
        );
        /*有电话时发送邮件*/
        Intent mEmailIntent = new Intent(android.content.Intent
                 .ACTION_SEND);
        mEmailIntent.setType("plain/text");
        /*设置收信人邮箱地址*/
        mEmailIntent.putExtra(android.content.Intent.EXTRA_EMAIL,
                 new String[]{mEditText01.toString()});
        /*设置邮件标题*/
        mEmailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT,
                 strEmailSubject);
        /*设置邮件内容*/
        mEmailIntent.putExtra(android.content.Intent.EXTRA_TEXT,
                 R.string.str_EmailBody+incomingNumber);
        /*实现发信处理*/
        startActivity(Intent.createChooser(mEmailIntent,
                 getResources().getString(R.string.str_message)));
        break;
      default:
        break;
    }
    super.onCallStateChanged(state, incomingNumber);
   }
  }
}
```

执行后当有电话进来时会显示提示信息，有短信时也会显示对应的提示，如图 10-13 所示。

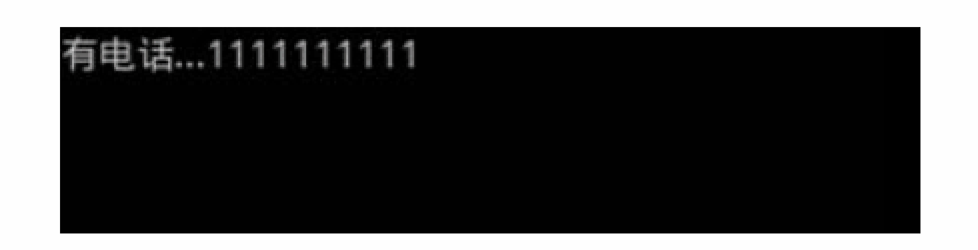


图 10-13　来电时的界面

10.1.4　调用内置邮件系统实现邮件发送

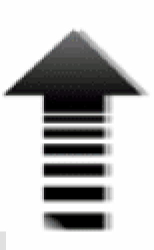
在使用 Intent 调用内置邮件系统时，使用的行为是 android.content.Intent. ACTION_SEND。实际上在 Android 系统中使用的邮件发送服务是调用 Gmail 程序，而并不是直接使用 SMTP 的 Protocol。

实　例	功　能	源码路径
实例 10-3	简易 E-mail 邮件发送系统	下载路径:\daima\10\sendEmail

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
  android:id="@+id/widget34"
```



```
android:layout width="fill parent"
android:layout_height="fill_parent"
android:background="@drawable/white"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
  android:id="@+id/myTextView1"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:text="@string/str_receive"
  android:layout x="60px"
  android:layout y="22px"
>
</TextView>
<TextView
  android:id="@+id/myTextView2"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:text="@string/str_cc"
  android:layout x="60px"
  android:layout y="82px"
>
</TextView>
<EditText
  android:id="@+id/myEditText1"
  android:layout width="fill parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"
  android:layout x="120px"
  android:layout y="12px"
>
</EditText>
<EditText
  android:id="@+id/myEditText2"
  android:layout_width="fill_parent"
  android:layout height="wrap content"
  android:textSize="18sp"
  android:layout x="120px"
  android:layout y="72px"
>
</EditText>
<Button
  android:id="@+id/myButton1"
  android:layout width="wrap content"
  android:layout height="124px"
  android:text="@string/str button"
  android:layout x="0px"
  android:layout y="2px"
>
```

```
  </Button>
  <TextView
    android:id="@+id/myTextView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/str_subject"
    android:layout_x="60px"
    android:layout_y="142px"
  >
  </TextView>
  <EditText
    android:id="@+id/myEditText3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="120px"
    android:layout_y="132px"
  >
  </EditText>
  <EditText
    android:id="@+id/myEditText4"
    android:layout_width="fill_parent"
    android:layout_height="209px"
    android:textSize="18sp"
    android:layout_x="0px"
    android:layout_y="202px"
  >
   </EditText>
</AbsoluteLayout>
```

(2) 编写主程序文件 sendEmailActivity.java，其具体实现流程如下。

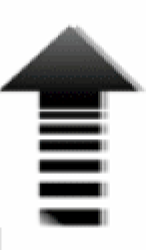
① 定义方法 boolean isEmail()用于判断用户输入的邮箱是否正确，具体代码如下：

```
public static boolean isEmail(String strEmail) {
    String strPattern = "^[a-zA-Z][\\w\\.-]*[a-zA-Z0-9]@[a-zA-Z0-9]
       [\\w\\.-]*[a-zA-Z0-9]\\.[a-zA-Z][a-zA-Z\\.]*[a-zA-Z]$";

    Pattern p = Pattern.compile(strPattern);
    Matcher m = p.matcher(strEmail);
    return m.matches();
   }
```

② 当用户长按文本框后，需通过 Content Provider 查找跳转到联系人中心，查找用户并返回邮箱，代码如下：

```
private OnLongClickListener searhEmail=new OnLongClickListener(){
       public boolean onLongClick(View arg0) {
           Uri uri=Uri.parse("content://contacts/people");
           Intent intent=new Intent(Intent.ACTION_PICK,uri);
           startActivityForResult(intent, PICK_CONTACT_SUBACTIVITY);
```



```
        return false;
        }
      ;
  };

  protected void onActivityResult(int requestCode, int resultCode,
     Intent data) {

     switch (requestCode) {
     case PICK CONTACT SUBACTIVITY:
        final Uri uriRet=data.getData();
        if(uriRet!=null)
        {
           try {
              Cursor c=managedQuery(uriRet, null, null, null, null);
              c.moveToFirst();
              //取得联系人的姓名
              String strName=c.getString(c.getColumnIndexOrThrow
                 (People.NAME));
              //取得联系人的 E-mail
              String[] PROJECTION=new String[]{
                    Contacts.ContactMethods. ID,
                    Contacts.ContactMethods.KIND,
                    Contacts.ContactMethods.DATA
              };
              //查询指定人的 E-mail
               Cursor newcur=managedQuery(
                     Contacts.ContactMethods.CONTENT URI,
                     PROJECTION,
                     Contacts.ContactMethods.PERSON ID+"=\'"
                     +c.getLong(c.getColumnIndex(People. ID))+"\'",
                     null, null);
              startManagingCursor(newcur);
              String email="";
              if(newcur.moveToFirst())
              {
                 email=newcur.getString(newcur.getColumnIndex
                        (Contacts.ContactMethods.DATA));
                 myEditText.setText(email);
              }

           } catch (Exception e) {
              // TODO: handle exception
              Toast.makeText(sendEmailActivity.this, e.toString(),
                 1000).show();
           }
        }
        break;

     default:
        break;
```

```
        }
        super.onActivityResult(requestCode, resultCode, data);
    };
```

要想实现跳转并取值返回，需要用到 startActivityForResult(intent,requestCode)，其中 requestCode 表示一个 Activity 要返回值的依据，可以是任意的 int 类型。我们可以自己定义常量，也可以自己指定数字。在程序中覆盖了 onActivityResult()方法，当程序收到 result 后，再重新加载写回原本需要加载的控件上。在本例中调用了文本框的长按事件，当文本框长按即自行跳转到联系人页面上，点击需要的联系人名称，返回该联系人的邮箱号回到主程序窗口并加载到文本上。

③ 单击“发送”按钮后触发事件开始发送。邮件发送程序并不复杂，主要是在 EditText 、Button 控件的构建，通过构造一个自定义的 Intent(android.content.Intent.ACTION_SEND)作为传送 E-mail 的 Activity 之用。在该 Intent 中，还必须使用 setType()来决定 E-mail 的格式，使用 putExtra()来置入寄件人(EXTRA_EMAIL)、主题(EXTRA_SUBJECT)、邮件内容(EXTRA_TEXT)以及其他 E-mail 的字段(EXTRA_BCC、EXTRA_CC)。对应的代码如下：

```
        myButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                Intent mailIntent=new Intent(android.content.Intent.ACTION SEND);
                mailIntent.setType("plain/test");
                strEmailReciver=new String[]{ myEditText.getText().toString() };
                strEmailCC=new String[]{myEditText2.getText().toString()};
                strEmailSubject=myEditText3.getText().toString();
                strEmailBody=myEditText4.getText().toString();
                mailIntent.putExtra(android.content.Intent.EXTRA EMAIL,
                    strEmailReciver);
                mailIntent.putExtra(android.content.Intent.EXTRA_CC, strEmailCC);
                mailIntent.putExtra(android.content.Intent.EXTRA SUBJECT,
                    strEmailSubject);
                mailIntent.putExtra(android.content.Intent.EXTRA TEXT,
                    strEmailBody);
            startActivity(Intent.createChooser(mailIntent, getResources().
                getString(R.string.send)));
            }
        });
```

(3) 在文件 AndroidManifest.xml 中声明权限，即当使用 Content Provider 查找联系人时必须在此配置文件中声明如下权限：

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

执行后的效果如图 10-14 所示。



图 10-14　执行效果

10.1.5　调用内置Gmail发送邮件

在本实例中自定义了一个 Intent，使用 Android.content.Intent.ACTION_SEND 参数通过手机寄发 E-mail 的服务，整个过程比较简单。在具体实现上，邮件的收发过程是通过 Android 内置的 Gmail 程序实现的，而并不是使用 SMTP 的 Protocol。为了确保邮件能够发出，必须在收件人字段上输入标准的邮件地址格式，如果格式不规范，则发送按钮处于不可用状态。

实　例	功　能	源码路径
实例 10-4	调用内置 Gmail 发送邮件	下载路径:\daima\10\DGmail

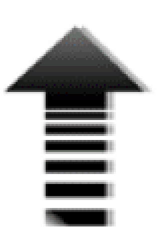
本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下：

```
<TextView
  android:id="@+id/myTextView1"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:text="@string/str receive"
  android:layout x="60px"
  android:layout y="22px"
>
</TextView>
<TextView
  android:id="@+id/myTextView2"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:text="@string/str cc"
  android:layout_x="60px"
```

```
    android:layout_y="82px"
  >
  </TextView>
  <EditText
    android:id="@+id/myEditText1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="120px"
    android:layout_y="12px"
  >
  </EditText>
  <EditText
    android:id="@+id/myEditText2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="120px"
    android:layout_y="72px"
  >
  </EditText>

  <TextView
    android:id="@+id/myTextView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/str_subject"
    android:layout_x="60px"
    android:layout_y="142px"
  >
  </TextView>
  <EditText
    android:id="@+id/myEditText3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="120px"
    android:layout_y="132px"
  >
  </EditText>
  <EditText
    android:id="@+id/myEditText4"
    android:layout_width="fill_parent"
    android:layout_height="209px"
    android:textSize="18sp"
    android:layout_x="0px"
    android:layout_y="202px"
  >
   </EditText><Button android:id="@+id/myButton1"
     android:layout_width="wrap_content" android:layout_height="124px"
     android:text="@string/str_button" android:layout_x="0px"
```



```
    android:layout y="2px">
  </Button>
```

(2) 编写主程序文件 GMail.java，其具体实现流程如下。

① 引用 content.Intent 类打开 E-mail 客户端，具体代码如下：

```
package irdc.GMail;

import irdc.GMail.R;

import java.util.regex.Matcher;
import java.util.regex.Pattern;
import android.app.Activity;
/*必须引用 content.Intent 类来打开 E-mail client*/
import android.content.Intent;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
```

② 分别声明四个 EditText、一个 Button 以及四个 String 变量，用于输入邮箱地址、邮件主体、副本和主题，具体代码如下：

```
public class GMail extends Activity
{
  /*声明四个 EditText、一个 Button 以及四个 String 变量*/
  private EditText mEditText01;
  private EditText mEditText02;
  private EditText mEditText03;
  private EditText mEditText04;
  private Button mButton01;
  private String[] strEmailReciver;
  private String strEmailSubject;
  private String[] strEmailCc;
  private String strEmailBody ;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*通过 findViewById 构造器来建构 Button 对象*/
    mButton01 = (Button)findViewById(R.id.myButton1);
    /*通过 findViewById 构造器来构造所有 EditText 对象*/
    mButton01.setEnabled(false);
    /*设置 OnKeyListener，当 key 事件发生时进行响应*/
    mEditText01 = (EditText)findViewById(R.id.myEditText1);
    mEditText02 = (EditText)findViewById(R.id.myEditText2);
    mEditText03 = (EditText)findViewById(R.id.myEditText3);
    mEditText04 = (EditText)findViewById(R.id.myEditText4);
```

③ 定义 setOnKeyListener 方法，如果用户输入为正规的 E-mail 文字，则按钮可用，反之则按钮不可用，具体代码如下：

```
/*若用户输入为正规 E-mail 文字，则按钮可用，反之则按钮不可用*/
mEditText01.setOnKeyListener(new EditText.OnKeyListener()
{
  @Override
  public boolean onKey(View v, int keyCode, KeyEvent event)
  {
    // TODO Auto-generated method stub
    /*如果是邮件地址格式，则按钮可按下*/
    if(isEmail(mEditText01.getText().toString()))
    {
      mButton01.setEnabled(true);
    }
    else
    {
      mButton01.setEnabled(false);
    }
    return false;
  }
});
```

④ 定义 onClickListener 响应按钮，当单击按钮后实现邮件发送处理。具体代码如下：

```
/*定义 onClickListener 响应按钮*/
mButton01.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub
    Intent mEmailIntent = new Intent(android.content.Intent.ACTION SEND);
    mEmailIntent.setType("plain/text");

    strEmailReciver = new String[]{mEditText01.getText().toString()};
    strEmailCc = new String[]{mEditText02.getText().toString()};
    strEmailSubject = mEditText03.getText().toString();
    strEmailBody = mEditText04.getText().toString();

    mEmailIntent.putExtra(android.content.Intent.EXTRA EMAIL, strEmailReciver);
    mEmailIntent.putExtra(android.content.Intent.EXTRA CC, strEmailCc);
    mEmailIntent.putExtra(android.content.Intent.EXTRA SUBJECT,
      strEmailSubject);
    mEmailIntent.putExtra(android.content.Intent.EXTRA TEXT, strEmailBody);
    startActivity(Intent.createChooser(mEmailIntent,
       getResources().getString(R.string.str message)));
  }
});
}
```

⑤ 定义 isEmail(String strEmail)方法，检查是否为规范的邮件地址格式。具体代码如下：



```
public static boolean isEmail(String strEmail)
{
  String strPattern = "^[a-zA-Z][\\w\\.-]*[a-zA-Z0-9]@[a-zA-Z0-9]
    [\\w\\.-]*[a-zA-Z0-9]\\.[a-zA-Z][a-zA-Z\\.]*[a-zA-Z]$";
  Pattern p = Pattern.compile(strPattern);
  Matcher m = p.matcher(strEmail);
  return m.matches();
}
}
```

执行后的效果如图 10-15 所示，输入手机号码，编写短信内容后，单击“发送”按钮即可完成短信发送功能，系统会提示信息成功，如图 10-16 所示。



图 10-15　执行效果



图 10-16　发信中提示

因为 Android 模拟器中没有内置 Gmail 客户端程序，所以当使用本实例发送邮件后，会显示“No application can perform this action”的提示。但是在现实手机设备上，如果运行本实例程序，会调用 Gmail 程序，成功实现邮件发送。

10.1.6　其他方法

还有其他使用 Intent 调用内置邮件程序发送邮件的方法，接下来将一一简要介绍。

(1) 直接设置具体的 E-mail 地址，例如通过下面的代码可以向地址为 aaa@gmail.com 的邮箱发送邮件。

```
Uri uri=Uri.parse("mailto:aaa@gmail.com");
Intent MymailIntent=new Intent(Intent.ACTION SEND,uri);
startActivity(MymailIntent);
```

(2) 在使用直接声明地址方式发送邮件时，可以实现群发功能，例如下面的代码同时向两个邮件地址发送邮件。

```
Intent testintent=new Intent(Intent.ACTION SEND);
String[] tos={"aaa@gmail.com"};
```

```
String[] ccs={"bbb@hotmail.com"};
testintent.putExtra(Intent.EXTRA_EMAIL, tos);
testintent.putExtra(Intent.EXTRA_CC, ccs);
testintent.putExtra(Intent.EXTRA_TEXT, "这是内容");
testintent.putExtra(Intent.EXTRA_SUBJECT, "这是标题");
testintent.setType("message/rfc822");
startActivity(Intent.createChooser(testintent, "发送"));
```

(3) 发送有附件的邮件。

方法 putExtra()的两个参数是 URL 地址，即在传递时第二个参数必须是 URI 格式的，那就意味着不能直接将图片传送过去，而是先要取得图片的 URI。在传送完毕后，E-mail 程序利用这个 URI 重新获取图片。为了获取 URI，需要先保存图片。例如下面的代码发送了附件为图片的邮件。

```
//创建保存文件
 String sdCardDir = Environment.getExternalStorageDirectory()+"/cameraApp/";
    File dirFile = new File(sdCardDir);
           if(!dirFile.exists()){
               dirFile.mkdir();
           }
    //创建保存文件
 bitmapFile = new File(dirFile,"Image"+picId+".jpg");
BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream
(bitmapFile));
Bm.compress(Bitmap.CompressFormat.JPEG, 80, bos);
bos.flush();
bos.close();
```

通过下面的代码向指定的地址发送了一幅图片。

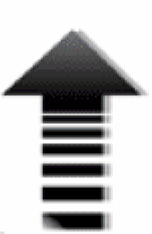
```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
Uri U=Uri.parse("file:///sdcard/logo.png");
emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL,"aaae@yahoo.com"
);
emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, "Test");
emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, "This is email's
message");
emailIntent.setType("image/png");
emailIntent.putExtra(android.content.Intent.EXTRA_STREAM,U);
startActivity(Intent.createChooser(emailIntent, "Email:"));
```

通过下面的代码发送了附件为音乐文件的邮件。

```
Intent testN=new Intent(Intent.ACTION_SEND);
  testN.putExtra(Intent.EXTRA_SUBJECT, "标题");
  testN.putExtra(Intent.EXTRA_STREAM, "file:///sdcard/music.mp3");
  startActivity(Intent.createChooser(testN, "发送"));
```

另外也可以将附件作为 File 对象来处理，例如下面的代码：

```
File file = new File("\sdcard\android123.cwj"); //附件文件地址
 Intent intent = new Intent(Intent.ACTION_SEND);
```



```
 intent.putExtra("subject", file.getName());
 intent.putExtra("body", "android123 - email sender");     //正文
 intent.putExtra(Intent.EXTRA STREAM, Uri.fromFile(file));
     //添加附件，附件为 file 对象
        if (file.getName().endsWith(".gz")) {
          intent.setType("application/x-gzip"); //如果是 gz 使用 gzip 的 mime
        } else if (file.getName().endsWith(".txt")) {
         intent.setType("text/plain"); //纯文本则用 text/plain 的 mime
        } else {
        intent.setType("application/octet-stream");
           //其他的均使用流当作二进制数据来发送
        }
startActivity(intent); //调用系统的 mail 客户端进行发送
```

10.2 使用 SmsManager 收发邮件

在 Android 系统中，除了可以使用 Intent 调用内置邮件系统发送邮件外，还可以使用类 SmsManager 来收发邮件。本节将详细讲解使用 SmsManager 实现邮件收发的基本知识。

10.2.1 SmsManager基础

在 Android 平台中，类 SmsManager 是用来管理短信服务操作的，例如发送数据、文本和数据单元等短信服务消息。可以通过调用 SmsManager.getDefault()来获取 SmsManager 对象。

1. SmsManager的常量

类 SmsManager 中的常量如下。

- public static final int RESULT_ERROR_GENERIC_FAILURE：表示普通错误，值为 1(0x00000001)。
- public static final int RESULT_ERROR_NO_SERVICE：表示服务当前不可用，值为 4 (0x00000004)。
- public static final int RESULT_ERROR_NULL_PDU：表示没有提供 pdu，值为 3 (0x00000003)。
- public static final int RESULT_ERROR_RADIO_OFF：表示无线广播被明确地关闭，值为 2 (0x00000002)。
- public static final int STATUS_ON_ICC_FREE：表示自由空间，值为 0 (0x00000000)。
- public static final int STATUS_ON_ICC_READ：表示接收且已读，值为 1 (0x00000001)。
- public static final int STATUS_ON_ICC_SENT：表示存储且已发送，值为 5 (0x00000005)。

- public static final int STATUS_ON_ICC_UNREAD：表示接收但未读，值为 3 (0x00000003)。
- public static final int STATUS_ON_ICC_UNSENT：表示存储但未发送，值为 7 (0x00000007)。

2. SmsManager的公有方法

1) ArrayList<String> divideMessage(String text)

功能：当短信超过 SMS 消息的最大长度时，将短信分割为几块。

参数：text，初始的消息，不能为空。

返回值：有序的 ArrayList<String>，可以重新组合为初始的消息。

2) static SmsManager getDefault()

功能：获取 SmsManager 的默认实例。

返回值：SmsManager 的默认实例。

3) void SendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)

功能：发送一个基于 SMS 的数据到指定的应用程序端口。

参数如下。

- destinationAddress：消息的目标地址。
- scAddress：服务中心的地址若为空，使用当前默认的 SMSC。
- destinationPort：消息的目标端口号。
- data：消息的主体，即消息要发送的数据。
- sentIntent：如果不为空，当消息成功发送或失败，这个 PendingIntent 就广播。如果代码是 Activity.RESULT_OK 表示成功，或 RESULT_ERROR_GENERIC_FAILURE、RESULT_ERROR_RADIO_OFF、RESULT_ERROR_NULL_PDU 之一表示错误。对应 RESULT_ERROR_GENERIC_FAILURE，sentIntent 可能包括额外的“错误代码”，包含一个无线电广播技术特定的值，通常只在修复故障时有用。每一个基于 SMS 的应用程序控制检测 sentIntent。如果 sentIntent 为空，调用者将检测所有未知的应用程序，这将导致在检测的时候发送较小数量的 SMS。
- deliveryIntent：如果不为空，当消息成功传送到接收者，这个 PendingIntent 就广播。

异常：如果 destinationAddress 或 data 是空时，抛出 IllegalArgumentException 异常。

4) void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliverIntents)

功能：发送一个基于 SMS 的多部分文本，调用者已经通过调用 divideMessage(String text)将消息分割成正确的大小。

参数如下。

- destinationAddress：消息的目标地址。
- scAddress：服务中心的地址若为空，使用当前默认的 SMSC。
- parts：有序的 ArrayList<String>，可以重新组合为初始的消息。
- sentIntents：与 SendDataMessage 方法中的含义一样，只不过这里指的是一组



PendingIntent。

- deliverIntents：与 SendDataMessage 方法中的含义一样，只不过这里指的是一组 PendingIntent。

异常：如果 destinationAddress 或 data 为空时，抛出 IllegalArgumentException 异常。

5) void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)

功能：发送一个基于 SMS 的文本。参数的意义和异常和前面的几个方法是一样的，在此不再赘述。

10.2.2　使用SmsManager发送短信

实　例	功　能	源码路径
实例 10-5	使用 SmsManager 发送短信	下载路径:\daima\10\jiandan

在本实例中，定义了两个 EditText 控件，分别用于获取收信人电话和短信正文，并设置判断手机号码规范化的方法和短信的字数不超过 70 个字符。

在具体实现上，是通过 SmsManage 对象的 sendTextMessage()方法来完成的。在 sendTextMessage()方法中要传入 5 个值，分别是：收件人地址 String、发送地址 String、正文 String、发送服务 PendingIntent 和送达服务 PendingIntent。本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下：

```
<TextView
  android:id="@+id/widget27"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:text="@string/str textview"
  android:textSize="16sp"
  android:layout x="0px"
  android:layout y="12px"
>
</TextView>
<EditText
  android:id="@+id/myEditText1"
  android:layout width="fill parent"
  android:layout height="wrap content"
  android:text=""
  android:textSize="18sp"
  android:layout x="60px"
  android:layout y="2px"
>
</EditText>
<EditText
  android:id="@+id/myEditText2"
  android:layout width="fill parent"
  android:layout height="223px"
  android:text=""
```




```
    android:textSize="18sp"
    android:layout x="0px"
    android:layout y="52px"
  >
  </EditText>
  <Button
    android:id="@+id/myButton1"
    android:layout width="162px"
    android:layout height="wrap content"
    android:text="@string/str button1"
    android:layout x="80px"
    android:layout y="302px"
  >
  </Button>
```

(2) 编写主程序文件 jiandan.java，其具体实现流程如下。

① 引用 dingIntent 类和 telephony.gsm.SmsManager 类，具体代码如下：

```
package irdc.jiandan;

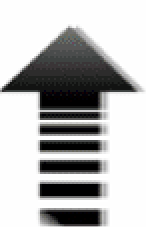
import android.app.Activity;
/*引用 PendingIntent 类才能使用 getBrocast()*/
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
/*引用 telephony.gsm.SmsManager 类才能使用 sendTextMessage()*/
import android.telephony.gsm.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import irdc.jiandan.R;

import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

② 声明变量：一个 Button 和两个 EditText。EditText 供获取输入收信人电话号码和短信内容，Button 用于激活发信处理程序。具体代码如下：

```
public class jiandan extends Activity
{
  /*声明变量：一个 Button 与两个 EditText*/
  private Button mButton1;
  private EditText mEditText1;
  private EditText mEditText2;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
```



```
/*
 * 通过 findViewById 构造器来建构
 * EditText1、EditText2 与 Button 对象
 */
mEditText1 = (EditText) findViewById(R.id.myEditText1);
mEditText2 = (EditText) findViewById(R.id.myEditText2);
mButton1 = (Button) findViewById(R.id.myButton1);

/*将默认文字加载到 EditText 中*/
mEditText1.setText("请输入号码");
mEditText2.setText("请输入内容!!");

/*设置 onClickListener 让用户点击 EditText 时做出反应*/
mEditText1.setOnClickListener(new EditText.OnClickListener()
{
  public void onClick(View v)
  {
    /*点击 EditText 时清空正文*/
    mEditText1.setText("");
  }
}
);
```

③ 设置 onClickListener()方法，用于响应用户点击 EditText 时做出反应。具体代码如下：

```
/*设置 onClickListener 让用户点击 EditText 时做出响应*/
mEditText2.setOnClickListener(new EditText.OnClickListener()
{
  public void onClick(View v)
  {
    /*点击 EditText 时清空正文*/
    mEditText2.setText("");
  }
}
);
```

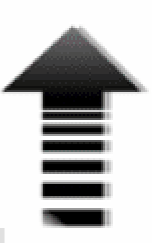
④ 设置 onClickListener 方法，功能是用户点击 Button 时做出响应，具体代码如下：

```
/*设置 onClickListener 让用户点击 Button 时做出响应*/
mButton1.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    /*由 EditText1 取得短信收件人电话*/
    String strDestAddress = mEditText1.getText().toString();
    /*由 EditText2 取得短信文字内容*/
    String strMessage = mEditText2.getText().toString();
    /*建构一取得 default instance 的 SmsManager 对象 */
    SmsManager smsManager = SmsManager.getDefault();
```

```
        // TODO Auto-generated method stub
```

⑤ 检查收件人电话格式与短信字数是否超过 70 字符，通过 smsManager.sendTextMessage 实现发送短信处理，具体实现代码如下：

```
        /*检查收件人电话格式与短信字数是否超过 70 字符*/
        if(isPhoneNumberValid(strDestAddress)==true &&
           iswithin70(strMessage)==true)
        {
          try
          {
            /*
            * 两个条件都检查通过的情况下，发送短信
            * 先建构一个 PendingIntent 对象并使用 getBroadcast()广播
            * 将 PendingIntent、电话、短信文字等参数
            * 传入 sendTextMessage()方法发送短信
            */
            PendingIntent mPI = PendingIntent.getBroadcast
            (jiandan.this, 0, new Intent(), 0);
            smsManager.sendTextMessage
            (strDestAddress, null, strMessage, mPI, null);
          }
          catch(Exception e)
          {
            e.printStackTrace();
          }
          Toast.makeText
          (
            jiandan.this,"送出成功!!" ,
            Toast.LENGTH SHORT
          ).show();
          mEditText1.setText("");
          mEditText2.setText("");
        }
        else
        {
          /* 电话格式与短信文字不符合条件时，以 Toast 提醒 */
          if (isPhoneNumberValid(strDestAddress)==false)
          { /*且字数超过 70 字符*/
            if(iswithin70(strMessage)==false)
            {
              Toast.makeText
              (
                jiandan.this,
                "电话号码格式错误+短信内容超过 70 字,请检查!!",
                Toast.LENGTH SHORT
              ).show();
            }
            else
            {
              Toast.makeText
```



```
          (
            jiandan.this,
            "电话号码格式错误,请检查!!" ,
            Toast.LENGTH SHORT
          ).show();
        }
      }
      /*字数超过 70 字符*/
      else if (iswithin70(strMessage)==false)
      {
        Toast.makeText
        (
          jiandan.this,
          "短信内容超过 70 字,请删除部分内容!!",
          Toast.LENGTH SHORT
        ).show();
      }
    }
  }
});

}
/*检查字符串是否为电话号码的方法, 并返回 true or false 的判断值*/
public static boolean isPhoneNumberValid(String phoneNumber)
{
  boolean isValid = false;
  /* 可接受的电话格式有:
   * ^\\(? : 可以使用“(”作为开头
   * (\\d{3}): 紧接着三个数字
   * \\)? : 可以使用“)”接续
   * [- ]? : 在上述格式后可以使用具选择性的“-”
   * (\\d{3}) : 再紧接着三个数字
   * [- ]? : 可以使用具选择性的“-”接续
   * (\\d{5})$: 以五个数字结束
   * 可以比较下列数字格式:
   * (123)456-7890、123-456-7890、1234567890、(123)-456-7890
  */
  String expression =
  "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{5})$";

  /* 可接受的电话格式有:
   * ^\\(? : 可以使用“(”作为开头
   * (\\d{3}): 紧接着三个数字
   * \\)? : 可以使用“)”接续
   * [- ]? : 在上述格式后可以使用具选择性的“-”
   * (\\d{4}) : 再紧接着四个数字
   * [- ]? : 可以使用具选择性的“-”接续
   * (\\d{4})$: 以四个数字结束
   * 可以比较下列数字格式:
   * (02)3456-7890、02-3456-7890、0234567890、(02)-3456-7890
  */
  String expression2=
```

```
      "^\\(?(\\d{3})\\)?[- ]?(\\d{4})[- ]?(\\d{4})$";

    CharSequence inputStr = phoneNumber;
    /*创建 Pattern*/
    Pattern pattern = Pattern.compile(expression);
    /*将 Pattern 以参数传入 Matcher 作 Regular expression*/
    Matcher matcher = pattern.matcher(inputStr);
    /*创建 Pattern2*/
    Pattern pattern2 =Pattern.compile(expression2);
    /*将 Pattern2 以参数传入 Matcher2 作 Regular expression*/
    Matcher matcher2= pattern2.matcher(inputStr);
    if(matcher.matches()||matcher2.matches())
    {
      isValid = true;
    }
    return isValid;
  }

  public static boolean iswithin70(String text)
  {
    if (text.length()<= 70)
    {
      return true;
    }
    else
    {
      return false;
    }
  }
}
```

在上述代码中，通过方法 PendingIntent.getBroadcast()自定义了 PendingIntent 并进行 Broadcast 广播，然后使用 SmsManager.getDefault()预先构建的 SmsManager 对象，并使用 sendTextMessage()方法将有关的数据以参数形式带入，这样即可完成发短信的任务。

执行后的效果如图 10-17 所示。输入手机号码，编写短信内容后，单击“发送”按钮即可完成短信发送功能。系统会提示成功信息，如图 10-18 所示。



图 10-17　执行效果



图 10-18　发送成功



如果短信内容和收信人号码格式不规范，会输出对应的错误提示。

10.2.3　解决Android邮件附件中文名乱码问题

邮件附件名的编码与邮件名的编码可以如出一辙。因为多功能 Internet 邮件扩充服务协议，即 MIME(Multipurpose Internet Mail Extensions)对附件名的规定是 US-ASCII(应该也是 ASCII)，所以该乱码 bug 的出现与 Java 和 Android 无关，是由于 MIME 的不规范。

邮件的标题用了一种更简短的格式来标注“字符编码”和“传输编码”。比如，标题内容为“中”，则在邮件源代码中表示为“=?GB2312?B?1tA=?= ”，其中第一个“=?”与“?”中间的部分指定了字符编码，在这个例子中指定的是 GB2312。“?”与“?”中间的“B”代表 Base64。如果是“Q”则代表 Quoted-Printable。最后“?”与“?=”之间的部分，就是经过 GB2312 转化成字节串，再经过 Base64 转化后的标题内容。如果“传输编码”改为 Quoted-Printable，同样，若标题内容为“中”，则：

```
// 正确的标题格式
Subject: =?GB2312?Q?=D6=D0?=
```

如果阅读邮件时出现乱码，一般是因为“字符编码”或“传输编码”指定有误，或者是没有指定。比如，有的发邮件组件在发送邮件时，标题“中”：

```
// 错误的标题格式
Subject: =?ISO-8859-1?Q?=D6=D0?=
```

这样的表示，实际上是明确指明了标题为 [0x00D6, 0x00D0]，即“ÖÐ”，而不是“中”。

根据上面的解释，那么解决乱码就没有问题了。在我们找到读取附件名的地方，如 mFileName，位于 Email/provider/EmailContent.java，首先对其进行 base64Encode 编码：String name = com.android.email.Utility.base64Encode(mFileName)，然后强制给 name 添加编码头和尾：String name2 = "=?utf8?B?" + name + "?="。这样接收邮件的客户端在检测到“=?utf8?B?”的时候，会对字符串进行 base64 和 UTF-8 的转码，乱码将不再出现。

10.3　使用包 commons-mail.jar 和 mail.jar

commons-mail.jar 和 mail.jar 是 Java 中两个比较重要的包，前者的功能是实现邮件发送，后者的功能是实现邮件接收。在 Android 系统应用中，可以使用这两个包来编写收发邮件程序。

10.3.1　使用commons-mail.jar发送邮件

对于包 commons-mail.jar 的使用方法，相信只要学过 Java 的读者都会有印象。本书将不再介绍它的用法，而直接用演示代码来讲解其用法。使用 commons-mail.jar 发送邮件的基本流程如下。

(1) 定义 mail 的一个 Bean，例如下面的代码：

```
public class Mail {
   private String toAddress;   // 邮件接收者
   private String nickname;    // 收件人昵称
   private String subject;     // 邮件主题
   private String content;     // 邮件内容
   private String ChartSet;    //字符集
    private Map<String, String> AttachmentsPath;   //附件路径列表
   /////setter() and getter()...
}
```

(2) 实现发送文本形式的邮件。设置接收者邮箱的信息参数，接下来就可以不断发送而无须再定义。例如下面的代码：

```
public class TextMailSender {

    private String hostname;
    private String username;
    private String password;
    private String address;
    private Boolean TLS;
    public TextMailSender(String hostname, String address, String
        username, String password, Boolean TLS) {
        this.hostname = hostname;
        this.username = username;
        this.password = password;
        this.address = address;
        this.TLS = TLS;
    }
    public void execute(Mail mail) throws EmailException{

        SimpleEmail email = new SimpleEmail();
       email.setTLS(TLS);
       email.setHostName(hostname);
       email.setAuthentication(username, password);    // 用户名和密码
       email.setFrom(address);                         // 发送地址

       email.addTo(mail.getToAddress());               // 接收地址
       email.setSubject(mail.getSubject());            // 邮件标题
       email.setCharset(mail.getChartSet());
       email.setMsg(mail.getContent());                // 邮件内容
       email.send();
    }
    public static void main(String[] args) {
        TextMailSender sender = new TextMailSender("smtp.qq.com",
            "cesul@qq.com", "cesul", "*******", true);

        Mail mail = new Mail();
        mail.setToAddress("cesul@qq.com");
        mail.setSubject("这又是一封测试邮件！");
```

```
        mail.setContent("呵呵呵呵呵");
        mail.setChartSet("utf-8");
        try {
            sender.execute(mail);
        } catch (EmailException e) {
            e.printStackTrace();
        }
        System.out.println("Finished");
    }
}
```

(3) 使用 commons-mail 定义另一个类，实现添加多个附件的功能。例如下面的演示代码：

```
public class AttachmentSender {

    private String hostname;  //"SMTP 服务器"
    private String username;
    private String password;
    private String address;
    private String nickname;
    private Boolean TLS;

    public AttachmentSender(String hostname, String address, String
        nickname, String username, String password, Boolean TLS) {
        this.hostname = hostname;
        this.nickname = nickname;
        this.username = username;
        this.password = password;
        this.address = address;
        this.TLS = TLS;
    }

    @SuppressWarnings("unchecked")
    public void execute(Mail mail) throws UnsupportedEncodingException,
        EmailException{
        // Create the email message
        MultiPartEmail email = new MultiPartEmail();
        email.setHostName(hostname);
        email.setAuthentication(username, password);
        email.setFrom(address, nickname);   //可以加入发信人称呼
        email.setTLS(TLS);

        email.setCharset(mail.getChartSet());
        email.addTo(mail.getToAddress(), mail.getNickname());
        email.setSubject(mail.getSubject());
        email.setMsg(mail.getContent());

        EmailAttachment attachment;
        //附件是多个，遍历
        Iterator<? extends Object> it = mail.getAttachmentsPath().
```

```
            entrySet().iterator();
        while (it.hasNext()) {
            Map.Entry<String, String> entry = (Map.Entry<String,
                String>)it.next();
            attachment = new EmailAttachment();
            attachment.setPath(entry.getKey());  //键是附件路径
            attachment.setDisposition(EmailAttachment.ATTACHMENT);
            attachment.setDescription(MimeUtility.encodeWord("附件","UTF-
                8",null));
            //值是附件描述名
            attachment.setName(MimeUtility.encodeWord(entry.getValue(),
               "UTF-8",null));
            email.attach(attachment);   // add the attachment
        }
        email.send();                       // 开始发送
    }


    public static void main(String[] args) {

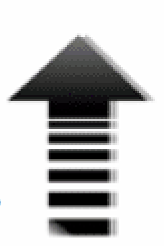
        AttachmentSender sender = new AttachmentSender("smtp.qq.com",
            "cesul@qq.com", "陈志钊", "cesul", "*******", true);

        Mail mail = new Mail();
        mail.setToAddress("cesul@qq.com");
        mail.setNickname("你好");
        mail.setSubject("Here is the picture you wanted");
        mail.setContent("呵呵呵呵呵");
        mail.setChartSet("utf-8");

        Map<String, String> attachment = new HashMap<String, String>();
        attachment.put("E:\\Photos\\2123.bmp", "这是你要的图片.bmp");
        attachment.put("E:\\Photos\\456.bmp", "这也是你要的图片.bmp");
            //不要把相同路径的文件发两次
        mail.setAttachmentsPath(attachment);

        try {
            sender.execute(mail);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (EmailException e) {
            e.printStackTrace();
        }
        System.out.println("Finished");
    }
}
```

经过上述流程，就实现了邮件发送功能。



10.3.2　使用mail.jar接收邮件

mail.jar 包比较强大，但是使用方法却比较麻烦。笔者本着无私奉献的精神和传道解惑的目的，决定将辛辛苦苦编写的实用资料无偿奉献给读者。希望读者以此为基础，既可以直接使用，也可以继续升级。

下面是通过 mail.jar 实现对邮件进行读取的代码。

```
package org.mail.core;

import java.io.*;
import java.text.*;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;

public class ReceiveMail {

    private MimeMessage mimeMessage = null;
    private String saveAttachPath = "";             // 附件下载后的存放目录
    private StringBuffer bodytext = new StringBuffer();
    // 存放邮件内容的 StringBuffer 对象
    private String dateformat = "yy-MM-dd HH:mm";  // 默认的日前显示格式
    /*构造函数，初始化一个 MimeMessage 对象*/
    public ReceiveMail() {
    }
    public ReceiveMail(MimeMessage mimeMessage) {
        this.mimeMessage = mimeMessage;
        System.out.println("create a ReceiveMail object........");
    }
    public void setMimeMessage(MimeMessage mimeMessage) {
        this.mimeMessage = mimeMessage;
    }
    /* 获得发件人的地址和姓名*/
    public String getFrom() throws Exception {
       InternetAddress address[] = (InternetAddress[]) mimeMessage.getFrom();
        String from = address[0].getAddress();
        if (from == null)
            from = "";
        String personal = address[0].getPersonal();
        if (personal == null)
            personal = "";
        String fromaddr = personal + "<" + from + ">";
        return fromaddr;
    }

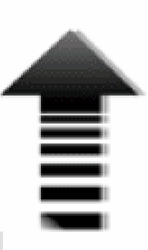
    /**
     * * 获得邮件的收件人、抄送和密送的地址和姓名，根据所传递参数的不同
     * “to”----收件人；“cc”---抄送人地址；
     * “bcc”---密送人地址
```

```
 */

public String getMailAddress(String type) throws Exception {
    String mailaddr = "";
    String addtype = type.toUpperCase();
    InternetAddress[] address = null;
    if (addtype.equals("TO") || addtype.equals("CC")
            || addtype.equals("BCC")) {
        if (addtype.equals("TO")) {
            address = (InternetAddress[]) mimeMessage
                    .getRecipients(Message.RecipientType.TO);
        } else if (addtype.equals("CC")) {
            address = (InternetAddress[]) mimeMessage
                    .getRecipients(Message.RecipientType.CC);
        } else {
            address = (InternetAddress[]) mimeMessage
                    .getRecipients(Message.RecipientType.BCC);
        }
        if (address != null) {
            for (int i = 0; i < address.length; i++) {
                String email = address[i].getAddress();
                if (email == null)
                    email = "";
                else {
                    email = MimeUtility.decodeText(email);
                }
                String personal = address[i].getPersonal();
                if (personal == null)
                    personal = "";
                else {
                    personal = MimeUtility.decodeText(personal);
                }
                String compositeto = personal + "<" + email + ">";
                mailaddr += "," + compositeto;
            }
            mailaddr = mailaddr.substring(1);
        }
    } else {
        throw new Exception("Error emailaddr type!");
    }
    return mailaddr;
}

/* 获得邮件主题*/

public String getSubject() throws MessagingException {
    String subject = "";
    try {
        subject = MimeUtility.decodeText(mimeMessage.getSubject());
        if (subject == null)
            subject = "";
```



```
        } catch (Exception exce) {
        }
        return subject;
    }

    /*获得邮件发送日期*/
    public String getSentDate() throws Exception {
        Date sentdate = mimeMessage.getSentDate();
        SimpleDateFormat format = new SimpleDateFormat(dateformat);
        return format.format(sentdate);
    }
    /*获得邮件正文内容*/
    public String getBodyText() {
        return bodytext.toString();
    }

    /*解析邮件，把得到的邮件内容保存到一个 StringBuffer 对象中，解析邮件。主要是根据
        MimeType 类型的不同执行不同的操作，一步一步地解析*/
    public void getMailContent(Part part) throws Exception {
        String contenttype = part.getContentType();
        int nameindex = contenttype.indexOf("name");
        boolean conname = false;
        if (nameindex != -1)
            conname = true;
        System.out.println("CONTENTTYPE: " + contenttype);
        if (part.isMimeType("text/plain") && !conname) {
            bodytext.append((String) part.getContent());
        } else if (part.isMimeType("text/html") && !conname) {
            bodytext.append((String) part.getContent());
        } else if (part.isMimeType("multipart/*")) {
            Multipart multipart = (Multipart) part.getContent();
            int counts = multipart.getCount();
            for (int i = 0; i < counts; i++) {
                getMailContent(multipart.getBodyPart(i));
            }
        } else if (part.isMimeType("message/rfc822")) {
            getMailContent((Part) part.getContent());
        } else {
        }
    }

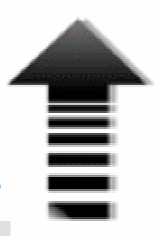
    /*判断此邮件是否需要回执，如果需要回执则返回 true，否则返回 false*/
    public boolean getReplySign() throws MessagingException {
        boolean replysign = false;
        String needreply[] = mimeMessage
                .getHeader("Disposition-Notification-To");
        if (needreply != null) {
            replysign = true;
        }
        return replysign;
    }
```




```
/*获得此邮件的Message-ID*/
public String getMessageId() throws MessagingException {
    return mimeMessage.getMessageID();
}

/*判断此邮件是否已读，如果未读则返回false，否则返回true*/
public boolean isNew() throws MessagingException {
    boolean isnew = false;
    Flags flags = ((Message) mimeMessage).getFlags();
    Flags.Flag[] flag = flags.getSystemFlags();
    System.out.println("flags's length: " + flag.length);
    for (int i = 0; i < flag.length; i++) {
        if (flag[i] == Flags.Flag.SEEN) {
            isnew = true;
            System.out.println("seen Message.......");
            break;
        }
    }
    return isnew;
}
/*判断此邮件是否包含附件*/
public boolean isContainAttach(Part part) throws Exception {
    boolean attachflag = false;
    String contentType = part.getContentType();
    if (part.isMimeType("multipart/*")) {
        Multipart mp = (Multipart) part.getContent();
        for (int i = 0; i < mp.getCount(); i++) {
            BodyPart mpart = mp.getBodyPart(i);
            String disposition = mpart.getDisposition();
            if ((disposition != null)
                    && ((disposition.equals(Part.ATTACHMENT)) ||
                    (disposition.equals(Part.INLINE))))
                    attachflag = true;
            else if (mpart.isMimeType("multipart/*")) {
                attachflag = isContainAttach((Part) mpart);
            } else {
                String contype = mpart.getContentType();
               if (contype.toLowerCase().indexOf("application") != -1)
                    attachflag = true;
                if (contype.toLowerCase().indexOf("name") != -1)
                    attachflag = true;
            }
        }
    } else if (part.isMimeType("message/rfc822")) {
        attachflag = isContainAttach((Part) part.getContent());
    }
    return attachflag;
}

/*保存附件*/
```



```
public void saveAttachMent(Part part) throws Exception {
    String fileName = "";
    if (part.isMimeType("multipart/*")) {
        Multipart mp = (Multipart) part.getContent();
        for (int i = 0; i < mp.getCount(); i++) {
            BodyPart mpart = mp.getBodyPart(i);
            String disposition = mpart.getDisposition();
            if ((disposition != null)
                    && ((disposition.equals(Part.ATTACHMENT)) ||
                    (disposition.equals(Part.INLINE)))) {
                fileName = mpart.getFileName();
                if (fileName.toLowerCase().indexOf("gb2312") != -1) {
                    fileName = MimeUtility.decodeText(fileName);
                }
                saveFile(fileName, mpart.getInputStream());
            } else if (mpart.isMimeType("multipart/*")) {
                saveAttachMent(mpart);
            } else {
                fileName = mpart.getFileName();
                if ((fileName != null)
                    && (fileName.toLowerCase().indexOf("GB2312") != -1)) {
                    fileName = MimeUtility.decodeText(fileName);
                    saveFile(fileName, mpart.getInputStream());
                }
            }
        }
    } else if (part.isMimeType("message/rfc822")) {
        saveAttachMent((Part) part.getContent());
    }
}

/*设置附件存放路径*/
public void setAttachPath(String attachpath) {
    this.saveAttachPath = attachpath;
}
/*设置日期显示格式*/
public void setDateFormat(String format) throws Exception {
    this.dateformat = format;
}

/*获得附件存放路径*/
public String getAttachPath() {
    return saveAttachPath;
}
/*真正地保存附件到指定目录里*/
private void saveFile(String fileName, InputStream in) throws Exception {
    String osName = System.getProperty("os.name");
    String storedir = getAttachPath();
    String separator = "";
    if (osName == null)
        osName = "";
```




```
        if (osName.toLowerCase().indexOf("win") != -1) {
            separator = "//";
            if (storedir == null || storedir.equals(""))
                storedir = "c://tmp";
        } else {
            separator = "/";
            storedir = "/tmp";
        }
        File storefile = new File(storedir + separator + fileName);
        System.out.println("storefile's path: " + storefile.toString());
        BufferedOutputStream bos = null;
        BufferedInputStream bis = null;
        try {
            bos = new BufferedOutputStream(new FileOutputStream(storefile));
            bis = new BufferedInputStream(in);
            int c;
            while ((c = bis.read()) != -1) {
                bos.write(c);
                bos.flush();
            }
        } catch (Exception exception) {
            exception.printStackTrace();
            throw new Exception("文件保存失败!");
        } finally {
            bos.close();
            bis.close();
        }
    }
    /*ReceiveMail 类测试*/
    public static void main(String args[]) throws Exception {
        String host = "pop.163.com";
        String username = "demo";   //您的邮箱用户名
        String password = "******";//您的邮箱密码
        Properties props = new Properties();
        Session session = Session.getDefaultInstance(props, null);
        Store store = session.getStore("pop3");
        store.connect(host, username, password);
        Folder folder = store.getFolder("INBOX");
        folder.open(Folder.READ ONLY);
        Message message[] = folder.getMessages();
        System.out.println("Messages's length: " + message.length);
        ReceiveMail pmm = null;
        for (int i = 0; i < message.length; i++) {
            pmm = new ReceiveMail((MimeMessage) message[i]);
            System.out
                    .println("Message " + i + " subject: " + pmm.getSubject());
            System.out.println("Message " + i + " sentdate: "
                    + pmm.getSentDate());
            System.out.println("Message " + i + " replysign: "
                    + pmm.getReplySign());
            System.out.println("Message " + i + " hasRead: " + pmm.isNew());
```

```
            System.out.println("Message " + i + " containAttachment: "
                    + pmm.isContainAttach((Part) message[i]));
            System.out.println("Message " + i + " form: " + pmm.getFrom());
            System.out.println("Message " + i + " to: "
                    + pmm.getMailAddress("to"));
            System.out.println("Message " + i + " cc: "
                    + pmm.getMailAddress("cc"));
            System.out.println("Message " + i + " bcc: "
                    + pmm.getMailAddress("bcc"));
            pmm.setDateFormat("yy年MM月dd日 HH:mm");
            System.out.println("Message " + i + " sentdate: "
                    + pmm.getSentDate());
            System.out.println("Message " + i + " Message-ID: "
                    + pmm.getMessageId());
            pmm.getMailContent((Part) message[i]);
            System.out.println("Message " + i + " bodycontent: /r/n"
                    + pmm.getBodyText());
            pmm.setAttachPath("c://tmp//coffeecat1124");
            pmm.saveAttachMent((Part) message[i]);
        }
    }
}
```

10.3.3　Android中用commons-email.jar和mail.jar收发邮件

(1) 在 Android 中可以使用 commons-mail.jar 发送邮件，例如下面的代码：

```
import org.apache.commons.net.smtp.SMTP;
import org.apache.commons.net.smtp.SMTPClient;
import org.apache.commons.net.smtp.SMTPReply;
SMTPClient client = new SMTPClient( );
client.connect("www.discursive.com");
int response = client.getReplyCode( );
if( SMTPReply.isPositiveCompletion( response ) ) {
// Set the sender and the recipients
client.setSender( "tobrien@discursive.com" );
client.addRecipient( "president@whitehouse.gov" );
client.addRecipient( "vicepresident@whitehouse.gov" );
// Supply the message via a Writer
Writer message = client.sendMessageData( );
message.write( "Spend more money on energy research.  Thanks." );
message.close( );
// Send the message and print a confirmation
boolean success = client.completePendingCommand( );
if( success ) {
System.out.println( "Message sent" );
}
} else {
System.out.println( "Error communicating with SMTP server" );
}
```

```
client.disconnect( );
```

注意： 在 Android 中很容易出错，建议编写如下异常处理代码:

```
 Multipart multipart = null;
Object obj = message.getContent();
  if (obj instanceof Multipart) {
              multipart = (Multipart) obj;
   } else {
            this.sendJavascript("javascript:alert('有封邮件接收出错')");
              continue;
 }
```

(2) 在 Android 中可以使用 mail.jar 接收邮件，例如下面的代码：

```
import org.apache.commons.io.CopyUtils;
import org.apache.commons.io.IOUtils;
import org.apache. commons.net.pop3.POP3Client;
import org.apache.commons.net.pop3.POP3MessageInfo;
POP3Client client = new POP3Client( );
client.connect("www.discursive.com");
client.login("tobrien@discursive.com", "secretpassword");
POP3MessageInfo[] messages = client.listMessages( );
for (int i = 0; i < messages.length; i++) {
int messageNum = messages[i].number;
System.out.println( "************* Message number: " + messageNum );
Reader reader = client.retrieveMessage( messageNum );
System.out.println( "Message:\n" + IOUtils.toString( reader ) );
IOUtils.closeQuietly( reader );
}
client.logout( );
client.disconnect( );
```

Android

第11章

让网络和多媒体接轨

在移动手机应用中，多媒体是一个重要的应用领域。从严格意义上来讲，多媒体包含了屏保、图片、音频、视频和相机等应用。如果将多媒体和网络相结合，则能给用户带来更加绚丽的体验。本章将详细介绍 Android 系统中开发一个网络多媒体应用的基本知识。

11.1 MediaPlayer 视频技术详解

在 Android 系统中，通常使用 MediaPlayer 接口实现音频和视频的播放功能。本节将简要介绍 MediaPlayer 接口的基本知识，为读者步入本书后面知识的学习打下基础。

11.1.1 MediaPlayer基础

MediaPlayer 的功能比较强大，既可以播放音频，也可以播放视频，另外还可以通过 VideoView 来播放视频，虽然 VideoView 比 MediaPlayer 简单易用，但定制性不如用 MediaPlayer，要视情况进行选用。MediaPlayer 播放音频比较简单，但是要播放视频就需要 SurfaceView。 SurfaceView 比普通的自定义 View 更有绘图上的优势，它支持完全的 OpenGL ES 库。

MediaPlayer 能被用来控制音频/视频文件或流媒体的回放，可以在 VideoView 中找到关于如何使用该类中的方法的例子。使用 MediaPlayer 实现音/视频播放的基本步骤如下。

(1) 生成 MediaPlayer 对象，根据播放文件从不同的地方使用不同的生成方式(参考 MediaPlayer API 即可)。

(2) 得到 MediaPlayer 对象后，根据实际需要调用不同的方法，如 start()、storp()、pause()、release()等。

需要注意的是，在不需要播放的时候要及时释放掉与 MediaPlayer 对象相连接的播放文件，因为直接使用 MediaPlayer 对象一般都是进行音频播放。

11.1.2 MediaPlayer的状态

图 11-1 显示了一个 MediaPlayer 对象被支持的播放控制操作驱动的生命周期和状态。其中，椭圆代表 MediaPlayer 对象可能驻留的状态；弧线表示驱动 MediaPlayer 在各个状态之间迁移的播放控制操作。这里有两种类型的弧线：由一个箭头开始的弧代表同步方法调用；而以双箭头开始的弧线代表异步方法调用。

通过图 11-1 可以知道一个 MediaPlayer 对象有以下几种状态。

(1) 当一个 MediaPlayer 对象刚刚被用 new 操作符创建或是调用了 reset()方法后，它就处于 Idle 状态。当调用了 release()方法后，它就处于 End 状态。这两种状态之间是 MediaPlayer 对象的生命周期。

在一个新构建的 MediaPlayer 对象和一个调用了 reset()方法的 MediaPlayer 对象之间有一个微小的但是十分重要的差别。在处于 Idle 状态时，调用 getCurrentPosition()、getDuration()、getVideoHeight()、getVideoWidth()、setAudioStreamType(int)、setLooping(boolean)、setVolume(float, float)、pause()、start()、stop()、seekTo(int)、prepare() 或者 prepareAsync() 方法都是编程错误。当一个 MediaPlayer 对象刚被构建的时候，内部的播放引擎和对象的状态都没有改变，在这个时候调用以上的方法，框架将无法回调客户端程序注册的

OnErrorListener.onError()方法；但若这个 MediaPlayer 对象调用了 reset()方法之后，再调用以上的方法，内部的播放引擎就会回调客户端程序注册的 OnErrorListener.onError()方法了，并将错误状态传入。

图 11-1　MediaPlayer对象

笔者在此建议，一旦一个 MediaPlayer 对象不再被使用，应立即调用 release()方法来释放在内部的播放引擎中与这个 MediaPlayer 对象关联的资源。资源可能包括如硬件加速组件的单态组件，若没有调用 release()方法可能会导致之后的 MediaPlayer 对象实例无法使用这种单态硬件资源，从而退回到软件实现或运行失败。一旦 MediaPlayer 对象进入了 End 状态，它不能再被使用，也没有办法再迁移到其他状态。

此外，在使用 new 操作符创建的 MediaPlayer 对象处于 Idle 状态时，而那些通过重载的 create()方法创建的 MediaPlayer 对象却并非处于 Idle 状态。事实上，如果成功调用了重载的 create()方法，那么这些对象已经是 Prepare 状态了。

(2) 在一般情况下，由于种种原因一些播放控制操作可能会失败，如不支持的音频/视频格式、缺少隔行扫描的音频/视频、分辨率太高、流超时等原因，等等。因此，错误报告和恢复在这种情况下是非常重要的。有时，由于编程错误，在处于无效状态的情况下可能发生了调用一个播放控制操作。在所有这些错误条件下，内部的播放引擎会调用一个由客户端程序提供的 OnErrorListener.onError()方法。客户端程序可以通过调用 MediaPlayer.setOnErrorListener(android.media.MediaPlayer.OnErrorListener)方法来注册 OnErrorListener。

如果一旦发生错误，MediaPlayer 对象会进入 Error 状态。为了重用一个处于 Error 状态的 MediaPlayer 对象，可以调用 reset()方法把这个对象恢复成 Idle 状态。注册一个 OnErrorListener 来获知内部播放引擎发生的错误是一个良好的编程习惯。在不合法的状态下调用一些方法，如 prepare()、prepareAsync()和 setDataSource()会抛出 IllegalStateException 异常。

(3) 调用 setDataSource(FileDescriptor)、setDataSource(String)、setDataSource(Context，Uri)或 setDataSource(FileDescriptor,long,long)方法会使处于 Idle 状态的对象迁移到 Initialized 状态。

若此时 MediaPlayer 处于其他的状态，调用 setDataSource()方法会抛出 IllegalStateException 异常。好的编程习惯是不要疏忽了在调用 setDataSource()方法的时候可能会抛出 IllegalArgumentException 异常和 IOException 异常。

(4) 在开始播放之前，MediaPlayer 对象必须要进入 Prepared 状态。

有以下两种方法(同步和异步)可以使 MediaPlayer 对象进入 Prepared 状态。

- prepare()方法(同步)：调用该方法返回则表示 MediaPlayer 对象已经进入了 Prepared 状态。
- prepareAsync()方法(异步)：调用该方法会使此 MediaPlayer 对象进入 Preparing 状态并返回，而内部的播放引擎会继续未完成的准备工作。

当同步版本返回时或异步版本的准备工作完全完成时就会调用客户端程序提供的 OnPreparedListener.onPrepared()监听方法。可以调用 MediaPlayer.setOnPreparedListener (android.media.MediaPlayer.OnPreparedListener)方法来注册 OnPreparedListener。

Preparing 是一个中间状态，在此状态下调用任何有影响的方法的结果都是未知的。在不合适的状态下调用 prepare()和 prepareAsync()方法会抛出 IllegalStateException 异常。当 MediaPlayer 对象处于 Prepared 状态时，可以调整音频/视频的属性，如音量、播放时是否一直亮屏、循环播放等。

(5) 在要开始播放时必须调用 start()方法。当此方法成功返回时，MediaPlayer 对象处于 Started 状态。isPlaying()方法可以被调用用来测试某个 MediaPlayer 对象是否处于 Started 状态。

当处于 Started 状态时，内部播放引擎会调用客户端程序提供的 OnBufferingUpdateListener.onBufferingUpdate()回调方法，此回调方法允许应用程序追踪流播放的缓冲状态。对一个已经处于 Started 状态的 MediaPlayer 对象调用 start()方法是没有影响的。

(6) 播放可以被暂停、停止以及调整当前播放位置。当调用 pause()方法并返回时，会使 MediaPlayer 对象进入 Paused 状态。注意 Started 与 Paused 状态的相互转换在内部的播



放引擎中是异步的。所以可能需要一点时间在 isPlaying()方法中更新状态，若在播放流内容，这段时间可能会有几秒。

调用 start()方法会让一个处于 Paused 状态的 MediaPlayer 对象从之前暂停的地方恢复播放。当调用 start()方法返回的时候，MediaPlayer 对象的状态又会变成 Started 状态。对一个已经处于 Paused 状态的 MediaPlayer 对象，pause()方法没有影响。

(7) 调用 stop()方法会停止播放，并且还会让一个处于 Started、Paused、Prepared 或 PlaybackCompleted 状态的 MediaPlayer 进入 Stopped 状态。对一个已经处于 Stopped 状态的 MediaPlayer 对象，stop()方法没有影响。

(8) 调用 seekTo()方法可以调整播放的位置。方法 seekTo(int)是异步执行的，所以它可以马上返回，但是实际的定位播放操作可能需要一段时间才能完成，尤其是播放流形式的音频/视频。当实际的定位播放操作完成之后，内部的播放引擎会调用客户端程序提供的 OnSeekComplete.onSeekComplete()回调方法。可以通过调用方法 setOnSeekCompleteListener (OnSeekCompleteListener)注册。

在此需要注意，seekTo(int)方法也可以在其他状态下调用，比如 Prepared、Paused 或 PlaybackCompleted 状态。此外，目前的播放位置，实际可以通过调用 getCurrentPosition()方法得到，它可以帮助如音乐播放器的应用程序不断更新播放进度。

(9) 当播放到流的末尾时完成播放。如果调用了 setLooping(boolean)方法开启了循环模式，那么 MediaPlayer 对象会重新进入 Started 状态。如果没有开启循环模式，那么内部的播放引擎会调用客户端程序提供的 OnCompletion.onCompletion()回调方法。可以通过调用 MediaPlayer.setOnCompletionListener(OnCompletionListener)方法来设置。内部的播放引擎一旦调用了 OnCompletion.onCompletion()回调方法，说明这个 MediaPlayer 对象进入了 PlaybackCompleted 状态。当处于 PlaybackCompleted 状态的时候，可以调用 start()方法来让这个 MediaPlayer 对象再进入 Started 状态。

11.1.3　MediaPlayer方法的有效状态和无效状态

MediaPlayer 方法的有效状态和无效状态如下。

- getCurrentPosition {Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted} {Error}：在有效状态成功调用该方法不会改变此时的状态，在无效状态调用该方法则会使该状态转换到错误状态中。
- getDuration {Prepared, Started, Paused, Stopped, PlaybackCompleted} {Idle, Initialized, Error}：在有效状态中成功调用该方法不会改变此时的状态，在无效状态调用该方法则会使该状态转换到错误状态中。
- getVideoHeight {Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted} {Error}：在有效状态成功调用该方法不会改变此时的状态，在无效状态调用该方法则会使该状态转换到错误状态中。
- getVideoWidth {Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted} {Error}：在有效状态成功调用该方法不会改变此时的状态，在无效状态调用该方法则会使该状态转换到错误状态中。

- isPlaying {Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted} {Error}：在有效状态成功调用该方法不会改变此时的状态，在无效状态调用该方法则会使该状态转换到错误状态中。
- pause {Started, Paused} {Idle, Initialized, Prepared, Stopped, PlaybackCompleted, Error}：在有效状态成功调用该方法则改变此时的状态到暂停状态，在无效状态调用该方法则会使该状态转换到错误状态中。
- prepare {Initialized, Stopped} {Idle, Prepared, Started, Paused, PlaybackCompleted, Error}：在有效状态成功调用该方法则改变此时的状态到准备状态，在无效状态调用该方法则会抛出错误状态异常。
- prepareAsync {Initialized, Stopped} {Idle, Prepared, Started, Paused, PlaybackCompleted, Error}：在有效状态成功调用该方法则改变此时的状态到准备状态，在无效状态调用该方法则会抛出错误状态异常。
- release any {}：在 release()该对象不再是可用的。
- reset {Idle, Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted, Error} {}：在 reset()后该对象如刚创建的一样。
- seekTo {Prepared, Started, Paused, PlaybackCompleted} {Idle, Initialized, Stopped, Error}：在有效状态成功调用该方法改变此时的状态到暂停状态，在无效状态调用该方法则会使该状态转换到错误状态中。
- setAudioStreamType {Idle, Initialized, Stopped, Prepared, Started, Paused, PlaybackCompleted} {Error} 在有效状态成功调用该方法改变此时的状态到暂停状态。
- setDataSource {Idle} {Initialized, Prepared, Started, Paused, Stopped, PlaybackCompleted, Error}：在有效状态成功调用该方法改变此时的状态到初始化状态，在无效状态调用该方法则会抛出错误状态异常。
- setDisplay any {}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- setLooping {Idle, Initialized, Stopped, Prepared, Started, Paused, PlaybackCompleted} {Error}：在有效状态成功调用该方法不会改变此时的状态，在无效状态调用该方法则会使该状态转换到错误状态中。
- isLooping any {}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- setOnBufferingUpdateListener any{}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- setOnCompletionListener any{}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- setOnErrorListener any{}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- setOnPreparedListener any{}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- setOnSeekCompleteListener any{}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- setScreenOnWhilePlaying any{}：在任何状态都可以调用该方法且不会改变当前对



象的状态。

- setVolume {Idle, Initialized, Stopped, Prepared, Started, Paused, PlaybackCompleted} {Error}：成功调用该方法不会改变当前的状态。
- setWakeMode any{}：在任何状态都可以调用该方法且不会改变当前对象的状态。
- start{Prepared, Started, Paused, PlaybackCompleted} {Idle, Initialized, Stopped, Error}：在有效状态成功调用该方法改变此时的状态到开始状态，在无效状态调用该方法则会转换到错误状态。
- stop {Prepared, Started, Stopped, Paused, PlaybackCompleted} {Idle, Initialized, Error}：在有效状态成功调用该方法改变此时的状态到停止状态，在无效状态调用该方法则会转换到错误状态。

11.1.4　MediaPlayer的接口

MediaPlayer 的接口如下。

- MediaPlayer.OnBufferingUpdateListener：该接口定义了调用指明网络上的媒体资源以缓冲流的形式播放。
- MediaPlayer.OnCompletionListener：该接口是为当媒体资源的播放完成后被调用的回放定义的。
- MediaPlayer.OnErrorListener：该接口定义了当在异步操作时(其他错误将会在呼叫方法的时候抛出异常)出现错误后调用的回放操作。
- MediaPlayer.OnInfoListener：该接口定义了与一些关于媒体或它的播放的信息以及/或者警告相关的被调用的回放。
- MediaPlayer.OnPreparedListener：该接口是为媒体的资源准备播放的时候调用回放定义的。
- MediaPlayer.OnSeekCompleteListener：该接口定义了指明查找操作完成后调用的回放操作。
- MediaPlayer.OnVideoSizeChangedListener：该接口定义了当视频大小被首次知晓或更新的时候调用的回放。

11.1.5　MediaPlayer的常量

MediaPlayer 的常量如下。

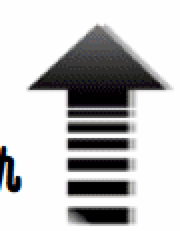
- int MEDIA_ERROR_NOT_VALID_FOR_PROGRESSIVE_PLAYBACK：视频流及其容器不支持连续的、非处于播放文件内的播放视频序列。
- int MEDIA_ERROR_SERVER_DIED：媒体服务终止。
- int MEDIA_ERROR_UNKNOWN：未指明的媒体播放错误。
- int MEDIA_INFO_BAD_INTERLEAVING：不正确的交叉存储技术意味着媒体被不适当地交叉存储或者根本就没有交叉存储。
- int MEDIA_INFO_METADATA_UPDATE：一套新的可用的元数据。
- int MEDIA_INFO_NOT_SEEKABLE：媒体位置不可查找。

- int MEDIA_INFO_UNKNOWN：未指明的媒体播放信息。
- int MEDIA_INFO_VIDEO_TRACK_LAGGING：视频对于解码器太复杂，以至于不能解码足够快的帧率。

11.1.6 MediaPlayer的公共方法

MediaPlayer 的公共方法如下。

- static MediaPlayer create(Context context, Uri uri)：指定从 URI 对应的资源文件中来装载音乐文件，并返回 MediaPlyaer 对象。
- static MediaPlayer create(Context context, int resid)：指定从资源 ID 对应的资源文件中来装载音乐文件，并返回 MediaPlyaer 对象。
- static MediaPlayer create(Context context, Uri uri, SurfaceHolder holder)：指定从资源 ID 对应的资源文件中来装载音乐文件，同时指定了 SurfaceHolder 对象并返回 MediaPlyaer 对象。
- int getCurrentPosition()：获得当前播放的位置。
- int getDuration()：获得文件段。
- int getVideoHeight()：获得视频的高度。
- int getVideoWidth()：获得视频的宽度。
- boolean isLooping()：检查 MedioPlayer 处于循环与否。
- boolean isPlaying()：检查 MedioPlayer 是否在播放。
- void pause()：暂停播放。
- void prepare()：让播放器处于准备状态(同步的)。
- void prepareAsync()：让播放器处于准备状态(异步的)。
- void release()：释放与 MediaPlayer 相关的资源。
- void reset()：重置 MediaPlayer 到初始化状态。
- void seekTo(int msec)：搜寻指定的时间位置。
- void setAudioStreamType(int streamtype)：为 MediaPlayer 设定音频流类型。
- void setDataSource(String path)：设定使用的数据源(文件路径或 http/rtsp 地址)。
- void setDataSource(FileDescriptor fd, long offset, long length)：设定使用的数据源(filedescriptor)。
- void setDataSource(FileDescriptor fd)：设定使用的数据源(filedescriptor)。
- void setDataSource(Context context, Uri uri)：设定一个如 URI 内容的数据源。
- void setDisplay(SurfaceHolder sh)：设定播放该 Video 的媒体播放器的 SurfaceHolder。
- void setLooping(boolean looping)：设定播放器循环或是不循环。
- void setOnBufferingUpdateListener(MediaPlayer.OnBufferingUpdateListener listener)：注册一个当网络缓冲数据流变化的时候调用的播放事件。
- void setOnCompletionListener(MediaPlayer.OnCompletionListener listener)：注册一个当媒体资源在播放的时候到达终点时调用的播放事件。



- void setOnErrorListener(MediaPlayer.OnErrorListener listener)：注册一个当在异步操作过程中发生错误的时候调用的播放事件。
- void setOnInfoListener(MediaPlayer.OnInfoListener listener)：注册一个当有信息/警告出现的时候调用的播放事件。
- void setOnPreparedListener(MediaPlayer.OnPreparedListener listener)：注册一个当媒体资源准备播放的时候调用的播放事件。
- void setOnSeekCompleteListener(MediaPlayer.OnSeekCompleteListener listener)：注册一个当搜寻操作完成后调用的播放事件。
- void setOnVideoSizeChangedListener(MediaPlayer.OnVideoSizeChangedListener listener)：注册一个当视频大小知晓或更新后调用的播放事件。
- void setScreenOnWhilePlaying(boolean screenOn)：控制当视频播放发生时是否使用 SurfaceHolder 来保持屏幕。
- void setVolume(float leftVolume, float rightVolume)：设置播放器的音量。
- void setWakeMode(Context context, int mode)：为 MediaPlayer 设置低等级的电源管理状态。
- void start()：开始或恢复播放。
- void stop()：停止播放。

11.2　VideoView 技术详解

VideoView 的用法和其他 Widget 私用方法类似，在使用时必须先在 Layout XML 中定义 VideoView 属性，然后在程序中通过 findViewById()方法即可创建 VideoView 对象。VideoView 的最大用处是播放视频文件。VideoView 类可以从不同的来源(例如资源文件或内容提供器) 读取图像，计算和维护视频的画面尺寸以使其适用于任何布局管理器，并提供一些诸如缩放、着色之类的显示选项。本节将简要介绍 VideoView 技术的基本知识。

11.2.1　VideoView的构造函数

VideoView 有三个构造函数。

其中第一个构造函数的语法格式如下：

```
public VideoView (Context context)
```

通过上述函数可以创建一个默认属性的 VideoView 实例。参数 context 表示视图运行的应用程序上下文，通过它可以访问当前的主题、资源，等等。

第二个构造函数的语法格式如下：

```
public VideoView (Context context, AttributeSet attrs)
```

通过上述函数可以创建一个带有 attrs 属性的 VideoView 实例，其中参数的具体说明如下。

- Context：表示视图运行的应用程序上下文，通过它可以访问当前主题、资源，

等等。

- ❑ attrs：用于视图的 XML 标签属性集合。

第三个构造函数的语法格式如下：

```
public VideoView (Context context, AttributeSet attrs, int defStyle)
```

通过上述函数可以创建一个带有 attrs 属性，并且指定其默认样式的 VideoView 实例。参数的具体说明如下。

- ❑ context：视图运行的应用程序上下文，通过它可以访问当前主题、资源，等等。
- ❑ attrs：用于视图的 XML 标签属性集合。
- ❑ defStyle：应用到视图的默认风格。如果为 0，则不应用风格(包括当前主题中的)。该值可以是当前主题中的属性资源，或者是明确的风格资源 ID。

11.2.2 VideoView的公共方法

VideoView 中公共方法的具体说明如下。

(1) public boolean canPause ()：判断是否能够暂停播放视频。

(2) public boolean canSeekBackward ()：判断是否能够倒退。

(3) public boolean canSeekForward ()：判断是否能够快进。

(4) public int getBufferPercentage ()：获得缓冲区的百分比。

(5) public int getCurrentPosition ()：获得当前的位置。

(6) public int getDuration ()：获得所播放视频的总时间。

(7) public boolean isPlaying ()：判断是否正在播放视频。

(8) public boolean onKeyDown (int keyCode, KeyEvent event)：是 KeyEvent.Callback.onKeyMultiple() 的默认实现。如果视图可用并可按，当按下 KEYCODE_DPAD_CENTER 或 KEYCODE_ENTER 时执行视图的按下事件。

此函数参数的具体说明如下。

- ❑ keyCode：表示按下的键的、在 KEYCODE_ENTER 中定义的键盘代码。
- ❑ event：KeyEvent 对象，定义了按钮动作。

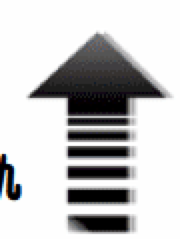
如果处理了事件则返回 true。如果允许下一个事件接收器处理该事件，则返回 false。

(9) public boolean onTouchEvent (MotionEvent ev)：通过该方法来处理触屏事件。参数 event 表示触屏事件。如果事件已经处理返回 true，否则返回 false。

(10) public boolean onTrackballEvent (MotionEvent ev)：利用该方法去处理轨迹球的动作事件，轨迹球相对于上次事件移动的位置能用 MotionEvent.getX() 和 MotionEvent.getY() 函数取回。对应用户按下一次方向键，它们通常作为一次移动处理(为了表现来自轨迹球的更小粒度的移动信息，它们返回小数)。参数 ev 表示动作的事件。

(11) public void pause ()：使得播放暂停。

(12) public int resolveAdjustedSize(int desiredSize, int measureSpec)：取得调整后的尺寸。如果 measureSpec 对象传入的模式是 UNSPECIFIED，那么返回的是 desiredSize。如果 measureSpec 对象传入的模式是 AT_MOST，返回的将是 desiredSize 和 measureSpec 对象的尺寸两者中最小的那个。如果 measureSpec 对象传入的模式是 EXACTLY，那么返回的是



measureSpec 对象中的尺寸大小值。

注意：MeasureSpec 是一个 android.view.View 内部类。它封装了从父类传送到子类的布局要求信息。每个 MeasureSpec 对象描述了控件的高度或者宽度。MeasureSpec 对象是由尺寸和模式组成的，有 3 个模式：UNSPECIFIED、EXACTLY、AT_MOST，这个对象由 MeasureSpec.makeMeasureSpec()函数创建。

(13) public void resume ()：用于恢复挂起的播放器。

(14) public void seekTo (int msec)：设置播放位置。

(15) public void setMediaController (MediaController controller)：设置媒体控制器。

(16) public void setOnCompletionListener (MediaPlayer.OnCompletionListener l)：注册在媒体文件播放完毕时调用的回调函数。

其中参数“l”表示要执行的回调函数。

(17) public void setOnErrorListener (MediaPlayer.OnErrorListener l)：注册在设置或播放过程中发生错误时调用的回调函数。如果未指定回调函数或回调函数返回假，则 VideoView 会通知用户发生了错误。其中参数“l”表示要执行的回调函数。

(18) public void setOnPreparedListener (MediaPlayer.OnPreparedListener l)：用于注册在媒体文件加载完毕，可以播放时调用的回调函数。其中参数“l”表示要执行的回调函数。

(19) public void setVideoPath (String path)：用于设置视频文件的路径名。

(20) public void setVideoURI (Uri uri)：设置视频文件的统一资源标识符。

(21) public void start ()：开始播放视频文件。

(22) public void stopPlayback ()：停止回放视频文件。

(23) public void suspend ()：挂起视频文件的播放。

11.3　在 Android 中播放网络上的 MP3

为了节约手机的存储空间，在听音乐时可以从网络上下载的方式播放 MP3。本节将通过一个具体实例的实现过程，来讲解在 Android 系统中播放网络 MP3 的方法。

实　例	功　能	源码路径
实例 11-1	播放网络中的 MP3	下载路径:\daima\11\mp

在本实例中，首先插入 4 个按钮，分别用于播放、暂停、重新播放和停止处理。执行后，通过 Runnable 发起运行线程，在线程中远程下载指定的 MP3 文件，是通过网络传输方式下载的。下载完毕后，临时保存到 SD 卡中，这样可以通过 4 个按钮对其进行控制。当程序关闭后，删除 SD 卡中的临时文件。本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，在里面插入 4 个图片按钮，主要代码如下：

```
<TextView
  android:id="@+id/myTextView1"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
```

```
  android:textColor="@drawable/blue"
  android:text="@string/hello"
/>
<LinearLayout
  android:orientation="horizontal"
  android:layout height="wrap content"
  android:layout width="fill parent"
  android:padding="10dip"
>
<ImageButton android:id="@+id/play"
  android:layout height="wrap content"
  android:layout width="wrap content"
  android:src="@drawable/play"
/>
<ImageButton android:id="@+id/pause"
  android:layout height="wrap content"
  android:layout width="wrap content"
  android:src="@drawable/pause"
/>
<ImageButton android:id="@+id/reset"
  android:layout height="wrap content"
  android:layout width="wrap content"
  android:src="@drawable/reset"
/>
<ImageButton android:id="@+id/stop"
  android:layout height="wrap content"
  android:layout width="wrap content"
  android:src="@drawable/stop"
/>
</LinearLayout>
```

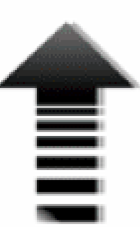
(2) 编写主程序文件 mp.java，其具体实现流程如下。

① 定义 currentFilePath 用于记录当前正在播放 MP3 的 URL 地址，定义 currentTempFilePath 表示当前播放 MP3 的路径。具体代码如下：

```
/*记录当前正在播放 MP3 的地址 URL*/
private String currentFilePath = "";
/*当前播放 MP3 的路径*/
private String currentTempFilePath = "";
private String strVideoURL = "";
```

② 使用 strVideoURL 设置要播放 MP3 文件的网址，并设置透明度。具体代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  /* MP3 文件不会被下载到 local*/
  strVideoURL = "http://www.lrn.cn/zywh/xyyy/yyxs/200805/
     W020080505536315331317.mp3";
  mTextView01 = (TextView)findViewById(R.id.myTextView1);
  /*设置透明度*/
```



```
getWindow().setFormat(PixelFormat.TRANSPARENT);
mPlay = (ImageButton)findViewById(R.id.play);
mReset = (ImageButton)findViewById(R.id.reset);
mPause = (ImageButton)findViewById(R.id.pause);
mStop = (ImageButton)findViewById(R.id.stop);
```

③ 编写单击“播放”按钮所触发的处理事件，具体代码如下：

```
/* 播放按钮 */
mPlay.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    /* 调用播放影片 Function */
    playVideo(strVideoURL);
    mTextView01.setText
    (
      getResources().getText(R.string.str play).toString()+
      "\n"+ strVideoURL
    );
  }
});
```

④ 编写单击“重播”按钮所触发的处理事件，具体代码如下：

```
/* 重新播放 */
mReset.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    if(bIsReleased == false)
    {
      if (mMediaPlayer01 != null)
      {
        mMediaPlayer01.seekTo(0);
        mTextView01.setText(R.string.str play);
      }
    }
  }
});
```

⑤ 编写单击“暂停”按钮所触发的处理事件，具体代码如下：

```
/* 暂停播放 */
mPause.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    if (mMediaPlayer01 != null)
    {
      if(bIsReleased == false)
      {
        if(bIsPaused==false)
        {
```

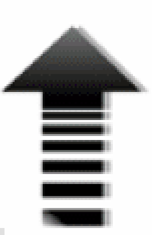
```
          mMediaPlayer01.pause();
          bIsPaused = true;
          mTextView01.setText(R.string.str_pause);
        }
        else if(bIsPaused==true)
        {
          mMediaPlayer01.start();
          bIsPaused = false;
          mTextView01.setText(R.string.str_play);
        }
      }
    }
  }
});
```

⑥ 编写单击“停止”按钮所触发的处理事件，具体代码如下：

```
  /*停止*/
  mStop.setOnClickListener(new ImageButton.OnClickListener()
  {
    public void onClick(View view)
    {
      try
      {
        if (mMediaPlayer01 != null)
        {
          if(bIsReleased==false)
          {
            mMediaPlayer01.seekTo(0);
            mMediaPlayer01.pause();
            //mMediaPlayer01.stop();
            //mMediaPlayer01.release();
            //bIsReleased = true;
            mTextView01.setText(R.string.str_stop);
          }
        }
      }
      catch(Exception e)
      {
        mTextView01.setText(e.toString());
        Log.e(TAG, e.toString());
        e.printStackTrace();
      }
    }
  });
}
```

⑦ 定义方法 playVideo(final String strPath)来播放指定的 MP3，其播放的是存储卡中暂时保存的 MP3 文件，具体代码如下：

```
private void playVideo(final String strPath)
{
  try
```



```
{
  if (strPath.equals(currentFilePath)&& mMediaPlayer01 != null)
  {
    mMediaPlayer01.start();
    return;
  }
  currentFilePath = strPath;
  mMediaPlayer01 = new MediaPlayer();
  mMediaPlayer01.setAudioStreamType(2);
```

⑧ 编写 setOnErrorListener 来监听错误处理，具体代码如下：

```
  /*错误事件 */
  mMediaPlayer01.setOnErrorListener(new MediaPlayer.OnErrorListener()
  {
    @Override
    public boolean onError(MediaPlayer mp, int what, int extra)
    {
      //TODO Auto-generated method stub
      Log.i(TAG, "Error on Listener, what: " + what + "extra: " + extra);
      return false;
    }
  });
```

⑨ 编写 setOnBufferingUpdateListener 来监听 MediaPlayer 缓冲区的更新，具体代码如下：

```
  /* 捕捉使用 MediaPlayer 缓冲区的更新事件 */
  mMediaPlayer01.setOnBufferingUpdateListener(new
      MediaPlayer.OnBufferingUpdateListener()
  {
    @Override
    public void onBufferingUpdate(MediaPlayer mp, int percent)
    {
      //TODO Auto-generated method stub
      Log.i(TAG, "Update buffer: " + Integer.toString(percent)+ "%");
    }
  });
```

⑩ 编写 setOnCompletionListener 来监听播放完毕所触发的事件，具体代码如下：

```
  /* 播放完毕所触发的事件 */
  mMediaPlayer01.setOnCompletionListener(new
MediaPlayer.OnCompletionListener()
  {
    @Override
    public void onCompletion(MediaPlayer mp)
    {
      //TODO Auto-generated method stub
      //delFile(currentTempFilePath);
      Log.i(TAG,"mMediaPlayer01 Listener Completed");
    }
  });
```

⑪ 编写 setOnPreparedListener 来监听开始阶段的事件，具体代码如下：

```
/* 开始阶段的监听 Listener */
mMediaPlayer01.setOnPreparedListener(new
    MediaPlayer.OnPreparedListener()
{
  @Override
  public void onPrepared(MediaPlayer mp)
  {
    //TODO Auto-generated method stub
    Log.i(TAG,"Prepared Listener");
  }
});
```

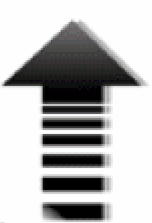
⑫ 将文件存储到 SD 卡后，通过方法 mMediaPlayer01.start()播放 MP3。具体代码如下：

```
  /* 用 Runnable 来确保文件在存储完毕后才开始 start() */
  Runnable r = new Runnable()
  {
    public void run()
    {
      try
      {
        /* setDataSource 将文件存到 SD 卡   */
        setDataSource(strPath);
        /*  因为线程顺利进行，所以在 setDataSource 后运行 prepare() */
        mMediaPlayer01.prepare();
        Log.i(TAG, "Duration: " + mMediaPlayer01.getDuration());

        /* 开始播放 mp3 */
        mMediaPlayer01.start();
        bIsReleased = false;
      }
      catch (Exception e)
      {
        Log.e(TAG, e.getMessage(), e);
      }
    }
  };
  new Thread(r).start();
}
```

⑬ 如果有异常则输出提示，具体代码如下：

```
catch(Exception e)
{
  if (mMediaPlayer01 != null)
  {
    /* 线程发生异常则停止播放   */
    mMediaPlayer01.stop();
    mMediaPlayer01.release();
  }
```



```
      e.printStackTrace();
    }
  }
```

⑭ 定义函数 setDataSource 用于存储 URL 的 MP3 文件到存储卡。首先判断传入的地址是否为 URL，然后创建 URL 对象和临时文件。具体代码如下：

```
/*  定义函数用于存储 URL 的 MP3 文件到存储卡  */
private void setDataSource(String strPath) throws Exception
{
  /*  判断传入的地址是否为 URL */
  if (!URLUtil.isNetworkUrl(strPath))
  {
    mMediaPlayer01.setDataSource(strPath);
  }
  else
  {
    if(bIsReleased == false)
    {
      /* 创建 URL 对象 */
      URL myURL = new URL(strPath);
      URLConnection conn = myURL.openConnection();
      conn.connect();

      /*  获取 URLConnection 的 InputStream */
      InputStream is = conn.getInputStream();
      if (is == null)
      {
        throw new RuntimeException("stream is null");
      }
      /*  创建临时文件  */
      File myTempFile = File.createTempFile("yinyue", "."+
          getFileExtension(strPath));
      currentTempFilePath = myTempFile.getAbsolutePath();
      FileOutputStream fos = new FileOutputStream(myTempFile);
      byte buf[] = new byte[128];
      do
      {
        int numread = is.read(buf);
        if (numread <= 0)
        {
          break;
        }
        fos.write(buf, 0, numread);
      }while (true);

      /*直到 fos 存储完毕，调用 MediaPlayer.setDataSource */
      mMediaPlayer01.setDataSource(currentTempFilePath);
      try
      {
        is.close();
```

```
        }
        catch (Exception ex)
        {
          Log.e(TAG, "error: " + ex.getMessage(), ex);
        }
      }
    }
  }
```

⑮ 定义方法 getFileExtension(String strFileName)来获取音乐文件的扩展名，如果无法顺利获取扩展名，则默认为“.dat”。具体代码如下：

```
  /*  获取音乐文件扩展名自定义函数 */
  private String getFileExtension(String strFileName)
  {
    File myFile = new File(strFileName);
    String strFileExtension=myFile.getName();

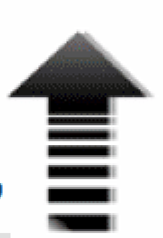
strFileExtension=(strFileExtension.substring(strFileExtension.lastIndexO
f(".")+1)).toLowerCase();
    if(strFileExtension=="")
    {
      /* 如果无法顺利获取扩展名则默认为.dat */
      strFileExtension = "dat";
    }
    return strFileExtension;
  }
```

⑯ 定义方法 delFile(String strFileName)来设置当离开程序时删除临时音乐文件，具体代码如下：

```
  /* 离开程序时需要调用自定义函数删除临时音乐文件*/
  private void delFile(String strFileName)
  {
    File myFile = new File(strFileName);
    if(myFile.exists())
    {
      myFile.delete();
    }
  }

  @Override
  protected void onPause()
  {
    //TODO Auto-generated method stub

    /*  删除临时文件  */
    try
    {
      delFile(currentTempFilePath);
    }
    catch(Exception e)
```



```
    {
      e.printStackTrace();
    }
    super.onPause();
  }
}
```

执行后可以通过“播放”、“暂停”、“重新播放”和“停止”四个按钮来控制指定的 MP3 音乐，如图 11-2 所示。

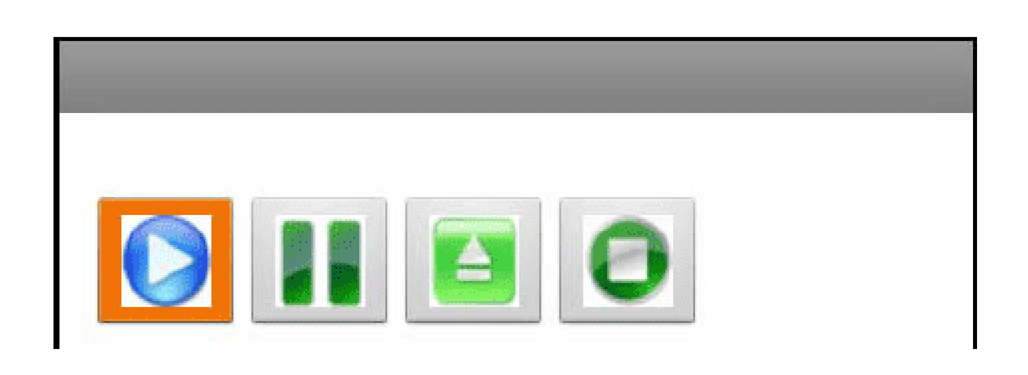


图 11-2　执行效果

11.4　在 Android 中下载在线铃声

在日常的手机应用中，我们经常从网络上下载一个 MP3 文件作为手机铃声。本节将通过一个具体实例的实现过程，来讲解在 Android 系统中下载在线铃声的方法。

题　目	目　的	源码路径
实例 11-2	下载在线铃声	下载路径:\daima\11\ling

在本实例中，我们可以在 EditText 中输入一个 MP3 的网址，当下载完成此网址的 MP3 后打开 RingtoneManager.ACTION_RINGTONE_PICKER 这个 Intent，在打开 Intent 的同时传入一个参数，这个 ACTION_RINGTONE_PICKER 的 Intent 会带入刚才下载的文件让用户选择。在具体实现过程中，会首先判断下载文件是否完整，并判断用户是否已设置铃声，会以 SD 卡中的铃声文件作为存储网络下载音乐文件的路径，打开 RingtoneManager 的 ACTION_RINGTONE_PICKER 的 Intent 让用户找到下载的音乐，并作为铃声。

本实例的主程序文件是 ling.java。具体实现流程如下。

(1) 用 private 声明系统中需要的对象，具体代码如下：

```
protected static final String APP_TAG = "DOWNLOAD_RINGTONE";
private Button mButton1;
private TextView mTextView1;
private EditText mEditText1;
private String strURL = "";
public static final int RINGTONE_PICKED = 0x108;
private String currentFilePath = "";
private String currentTempFilePath = "";
private String fileEx="";
private String fileNa="";
private String strRingtoneFolder = "/sdcard/music/ling";
```

(2) 判断是否包含文件夹“/sdcard/music/ringtones”，如果不存在则输出提示。具体代码如下：

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

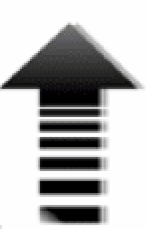
  mButton1 =(Button) findViewById(R.id.myButton1);
  mTextView1 = (TextView) findViewById(R.id.myTextView1);
  mEditText1 = (EditText) findViewById(R.id.myEditText1);
  /*判断是否有/sdcard/music/ringtones 文件夹*/
  if(bIfExistRingtoneFolder(strRingtoneFolder))
  {
    Log.i(APP_TAG, "Ringtone Folder exists.");
  }
```

(3) 使用 fileEx 和 getFile 取得远程 MP3 文件的名称，具体代码如下：

```
  mButton1.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View arg0)
    {
      strURL = mEditText1.getText().toString();
          Toast.makeText(ling.this, getString(R.string.str_msg)
          ,Toast.LENGTH_SHORT).show();
      /*取得文件名称*/
      fileEx = strURL.substring(strURL.lastIndexOf(".")+1,strURL.
           length()).toLowerCase();
      fileNa = strURL.substring(strURL.lastIndexOf("/")+1,strURL.
           lastIndexOf("."));
      getFile(strURL);
    }
   });
}
```

(4) 定义方法 getMIMEType(File f)来判断文件 MimeType 的打开格式，具体代码如下：

```
/* 判断文件 MimeType 的 method */
private String getMIMEType(File f)
{
  String type="";
  String fName=f.getName();
  /* 取得扩展名 */
  String end=fName.substring(fName.lastIndexOf(".")+1,
             fName.length()).toLowerCase();
  /* 依扩展名的类型决定 MimeType */
  if(end.equals("m4a")||end.equals("mp3")||end.equals("mid")||
      end.equals("xmf")||end.equals("ogg")||end.equals("wav"))
```



```
    {
      type = "audio";
    }
    else if(end.equals("3gp")||end.equals("mp4"))
    {
      type = "video";
    }
    else if(end.equals("jpg")||end.equals("gif")||
             end.equals("png")||end.equals("jpeg")||
              end.equals("bmp"))
    {
      type = "image";
    }
    else
    {
      type="*";
    }
    /*如果无法直接打开，就跳出软件列表供用户选择 */
    if(end.equals("image"))
    {
    }
    else
    {
      type += "/*";
    }
    return type;
  }
```

(5) 定义方法 getFile(final String strPath)来获取 MP3 文件，如果地址和当前地址一样则直接使用 getDataSource 数据，如果有异常则输出异常信息。具体代码如下：

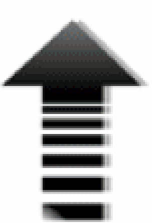
```
  private void getFile(final String strPath)
  {
    try
    {
      if (strPath.equals(currentFilePath) )
      {
       getDataSource(strPath);
      }
      currentFilePath = strPath;
      Runnable r = new Runnable()
      {
        public void run()
        {
          try
          {
            getDataSource(strPath);
          }
          catch (Exception e)
          {
            Log.e(APP_TAG, e.getMessage(), e);
```

```
        }
      }
    };
    new Thread(r).start();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
```

(6) 定义方法 getDataSource(String strPath)来获取远程文件，如果地址错误则输出错误信息。具体代码如下：

```
/*取得远程文件*/
private void getDataSource(String strPath) throws Exception
{
  if (!URLUtil.isNetworkUrl(strPath))
  {
    mTextView1.setText("错误的 URL");
  }
  else
  {
    /*取得 URL*/
    URL myURL = new URL(strPath);
    /*创建连接*/
    URLConnection conn = myURL.openConnection();
    conn.connect();
    /*InputStream 下载文件*/
    InputStream is = conn.getInputStream();
    if (is == null)
    {
      throw new RuntimeException("stream is null");
    }

    /*创建文件地址*/
    File myTempFile = new File("/sdcard/music/ling/",
         fileNa+"."+fileEx);
    /*取得在暂存盘的路径*/
    currentTempFilePath = myTempFile.getAbsolutePath();
    /*将文件写入暂存盘*/
    FileOutputStream fos = new FileOutputStream(myTempFile);
    byte buf[] = new byte[128];
    do
    {
      int numread = is.read(buf);
      if (numread <= 0)
      {
        break;
      }
      fos.write(buf, 0, numread);
    }while (true);
```



(7) 打开 RingtonManager 以选择铃声，通过 Intent 对象 intent 来设置铃声，然后设置显示铃声的文件夹和显示铃声开头。如果有异常则输出异常。具体代码如下：

```
    /* 打开 RingtonManager 进行铃声选择 */
    String uri = null;
    if(bIfExistRingtoneFolder(strRingtoneFolder))
    {
      /*设置铃声*/
      Intent intent = new Intent( RingtoneManager.
            ACTION RINGTONE PICKER);
      /*设置显示铃声的文件夹*/
      intent.putExtra( RingtoneManager.EXTRA RINGTONE TYPE,
            RingtoneManager.TYPE RINGTONE);
      /*设置显示铃声开头*/
      intent.putExtra( RingtoneManager.EXTRA RINGTONE TITLE,
            "设置铃声");
      if( uri != null)
      {
        intent.putExtra( RingtoneManager.
             EXTRA RINGTONE EXISTING URI, Uri.parse( uri));
      }
      else
      {
        intent.putExtra( RingtoneManager.
             EXTRA RINGTONE EXISTING URI, (Uri)null);
      }
      startActivityForResult(intent, RINGTONE PICKED);
    }

    try
    {
      is.close();
    }
    catch (Exception ex)
    {
      Log.e(APP TAG, "error: " + ex.getMessage(), ex);
    }
  }
}
```

(8) 定义方法 onActivityResult 根据用户选择的铃声设置保存对应的信息。当选择完毕后，再次返回选择 Activity 界面。具体代码如下：

```
protected void onActivityResult(int requestCode,
                  int resultCode, Intent data)
{
  if (resultCode != RESULT OK)
  {
    return;
  }
  switch (requestCode)
```

```
  {
    case (RINGTONE PICKED):
      try
      {
        Uri pickedUri = data.getParcelableExtra
        (RingtoneManager.EXTRA RINGTONE PICKED URI);
        if(pickedUri!=null)
        {
          RingtoneManager.setActualDefaultRingtoneUri
          (ling.this,RingtoneManager.TYPE RINGTONE,
              pickedUri);
        }
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
      break;
    default:
      break;
  }
  super.onActivityResult(requestCode, resultCode, data);
}
```

(9) 定义 bIfExistRingtoneFolder 来判断是否包含文件夹“/sdcard/music/ringtones”，具体代码如下：

```
/*判断是否包含“/sdcard/music/ringtones 文件夹”*/
private boolean bIfExistRingtoneFolder(String strFolder)
{
  boolean bReturn = false;

  File f = new File(strFolder);
  if(!f.exists())
  {
    /*创建/sdcard/music/ringtones 文件夹*/
    if(f.mkdirs())
    {
      bReturn = true;
    }
    else
    {
      bReturn = false;
    }
  }
  else
  {
    bReturn = true;
  }
  return bReturn;
}
}
```



执行后会先显示一个下载界面，单击“点击下载”按钮开始下载指定的 MP3 文件，如图 11-3 所示。下载完成后会弹出一个界面。选择一种选项，并单击 OK 按钮完成铃声设置。



图 11-3　执行效果

11.5　在 Android 中上传文件到远程服务器

文件上传对于广大读者来说并不陌生，在手机中我们同样可以实现网站上传功能。本节将通过一个具体实例的实现过程，介绍在 Android 中上传文件的基本流程。

实　例	功　能	源码路径
实例 11-3	上传文件到远程服务器	下载路径:\daima\11\chuan

(1) 编写布局文件 main.xml，主要代码如下：

```
<TextView
    android:id="@+id/myText1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="@string/str title"
    android:textSize="20sp"
    android:textColor="@drawable/black"
    android:layout x="10px"
    android:layout y="12px"
  >
  </TextView>
  <TextView
    android:id="@+id/myText2"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:textSize="16sp"
    android:textColor="@drawable/black"
    android:layout x="10px"
    android:layout y="52px"
  >
  </TextView>
  <TextView
    android:id="@+id/myText3"
    android:layout_width="wrap_content"
```

```
    android:layout height="wrap content"
    android:textSize="16sp"
    android:textColor="@drawable/black"
    android:layout x="10px"
    android:layout_y="102px"
  >
  </TextView>
  <Button
    android:id="@+id/myButton"
    android:layout width="92px"
    android:layout height="49px"
    android:text="@string/str button"
    android:textSize="15sp"
    android:layout x="90px"
    android:layout y="170px"
  >
  </Button>
```

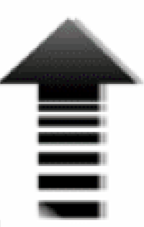
(2) 编写主程序文件 chuan.java，其具体实现流程如下。

① 分别声明变量 newName、uploadFile 和 actionUrl，具体代码如下：

```
public class chuan extends Activity
{
  /* 变量声明
   * newName：上传后在服务器上的文件名称
   * uploadFile：要上传的文件路径
   * actionUrl：服务器上对应的程序路径 */
  private String newName="image.jpg";
  private String uploadFile="/data/data/irdc.example9/image.jpg";
  private String actionUrl="http://127.127.0.1/upload/upload.jsp";
  private TextView mText1;
  private TextView mText2;
  private Button mButton;
```

② 通过 mText1 对象获取文件路径，根据 mText2 设置上传网址，单击按钮后调用上传方法 uploadFile()。具体代码如下：

```
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mText1 = (TextView) findViewById(R.id.myText2);
    mText1.setText("文件路径：\n"+uploadFile);
    mText2 = (TextView) findViewById(R.id.myText3);
    mText2.setText("上传网址：\n"+actionUrl);
    /* 设置 mButton 的 onClick 事件处理 */
    mButton = (Button) findViewById(R.id.myButton);
    mButton.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View v)
      {
        uploadFile();
```



```
    }
  });
}
```

③ 定义方法 uploadFile()将文件上传至 Server，具体代码如下：

```
/* 上传文件至 Server 的方法 */
private void uploadFile()
{
  String end = "\r\n";
  String twoHyphens = "--";
  String boundary = "*****";
  try
  {
    URL url =new URL(actionUrl);
    HttpURLConnection con=(HttpURLConnection)url.openConnection();
    /* 允许 Input、Output，不使用 Cache */
    con.setDoInput(true);
    con.setDoOutput(true);
    con.setUseCaches(false);
    /* 设置传送的 method=POST */
    con.setRequestMethod("POST");
    /* setRequestProperty */
    con.setRequestProperty("Connection", "Keep-Alive");
    con.setRequestProperty("Charset", "UTF-8");
    con.setRequestProperty("Content-Type",
         "multipart/form-data;boundary="+boundary);
    /* 设置 DataOutputStream */
    DataOutputStream ds =
      new DataOutputStream(con.getOutputStream());
    ds.writeBytes(twoHyphens + boundary + end);
    ds.writeBytes("Content-Disposition: form-data; " +
         "name=\"file1\";filename=\"" +
          newName +"\"" + end);
    ds.writeBytes(end);
    /* 取得文件的 FileInputStream */
    FileInputStream fStream = new FileInputStream(uploadFile);
    /* 设置每次写入 1024 字节 */
    int bufferSize = 1024;
    byte[] buffer = new byte[bufferSize];
    int length = -1;
    /* 从文件读取数据至缓冲区 */
    while((length = fStream.read(buffer)) != -1)
    {
      /* 将资料写入 DataOutputStream 中 */
      ds.write(buffer, 0, length);
    }
    ds.writeBytes(end);
    ds.writeBytes(twoHyphens + boundary + twoHyphens + end);
    fStream.close();
    ds.flush();
```

```
      /* 取得 Response 内容 */
      InputStream is = con.getInputStream();
      int ch;
      StringBuffer b =new StringBuffer();
      while( ( ch = is.read() ) != -1 )
      {
        b.append( (char)ch );
      }
      /* 将 Response 显示在对话框中 */
      showDialog(b.toString().trim());
      /* 关闭 DataOutputStream */
      ds.close();
    }
    catch(Exception e)
    {
      showDialog(""+e);
    }
  }
```

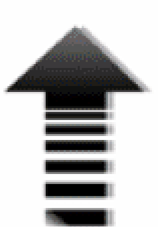
④ 定义方法 showDialog(String mess)来显示提示对话框，具体代码如下：

```
  /* 显示 Dialog 的方法*/
  private void showDialog(String mess)
  {
    new AlertDialog.Builder(example9.this).setTitle("Message")
     .setMessage(mess)
     .setNegativeButton("确定",new DialogInterface.OnClickListener()
     {
       public void onClick(DialogInterface dialog, int which)
       {
       }
     })
     .show();
  }
}
```

执行后单击“上传”按钮可以将指定的文件上传到服务器，如图 11-4 所示。



图 11-4　执行效果



11.6　在 Android 中开发一个远程下载系统

在 Android 系统中可以安装.APK 格式的文件，所以本节将通过一个具体实例的实现过程，讲解开发能够远程下载.APK 格式文件的过程。

11.6.1　基础知识介绍

APK 是 Android Package 的缩写，即 Android 安装包。APK 是类似 Symbian Sis 或 Sisx 的文件格式。通过将 APK 文件直接传到 Android 模拟器或 Android 手机中执行即可安装。

APK 文件和 Sis 一样最终把 Android SDK 编译的工程打包成一个安装程序文件，格式为 APK。APK 文件其实是 zip 格式，但后缀名被修改为 APK，通过 UnZip 解压后，可以看到 Dex 文件，Dex 是 Dalvik VM executes 的全称，即 Android Dalvik 执行程序，并非 Java ME 的字节码，而是 Dalvik 字节码。一个 APK 文件结构为：META-INF\Jar，此文件结构的具体说明如下。

- res\：存放资源文件的目录。
- AndroidManifest.xml：程序全局配置文件。
- classes.dex：Dalvik 字节码。
- resources.arsc：编译后的二进制资源文件。

Android 在运行一个程序时，首先需要 UnZip 解压缩，这一点和 Symbian 比较相似，而和 Windows Mobile 中的 PE 文件有所区别。这样做对于程序的保密性和可靠性不是很高，通过 dexdump 命令可以反编译，但这样做符合发展规律，微软的 Windows Gadgets 或者说 WPF 也采用了这种构架方式。在 Android 平台中 Dalvik vm 的执行文件被打包为 APK 格式，最终运行时加载器会解压然后获取编译后的 androidmanifest.xml 文件中的 permission 分支相关的安全访问，但仍然存在很多安全限制，如果你将 APK 文件传到 /system/app 文件夹下会发现执行是不受限制的。我们平时安装的文件可能不是这个文件夹，而在 Android ROOM 中系统的 APK 文件默认会放入该文件夹，它们具有 ROOT 权限。

1. 下载APK应用程序

我们可以从哪里取得好用的 Android APK 应用程序，并安装到 Android 手机上呢？对拥有 G1 实体手机的使用者而言，Android Market 就是最佳的地方，只要使用手机内应用程序列表的 Market 程序，就可以直接连接到 Android Market，而点选喜爱的应用程序后，就会直接下载并安装到 G1 手机上。不过对使用 Android 仿真器的使用者而言，就没有如此方便了，Android 仿真器并没有 Android Market 应用程序，只能使用内附的浏览器浏览 Android Market。为何说是浏览呢？因为 Android Market 不是采用通用网页浏览方式来下载文件，虽然可以使用常见的浏览器看到 Android Market 上的应用程序，但却没有办法下载到 Android 仿真器或一般的计算机上，而 Android Market 采用特有的网页 API，使用 Native UI 的方式来访问，只有通过内建在 G1 手机内的 Market 应用程序，才能下载 Android Market 网页中的应用程序，并自动安装到 G1 手机上。

所以 Android 仿真器的使用者，只好浏览该网页上的应用程序，然后通过搜索引擎去找找看有没有开发人员将应用程序放到 Android Market 之后，还另外将 APK 文件放置在一般网页上了。到此为止，使用 Android 仿真器的用户，也不需要这么灰心，因为有太多的人遇到同样的问题，也就生成非常多的 Android 应用程序网页，你可以浏览这些网页并把上面的 APK 文件下载到一般计算机上，再安装到 Android 仿真器上。

下面列出了常用的 APK 应用程序下载网站：

- http://andappstore.com/
- http://www.getjar.com/software
- http://www.phoload.com/android
- http://slideme.org/
- http://androidforums.com/market/
- http://www.cyrket.com/
- http://www.androidfreeware.org/
- http://androidsoftwaredownload.com/
- http://www.freeandroidsoft.com/
- http://code.google.com/p/apps-for-android/
- http://code.google.com/p/openintents/downloads/list

2. 安装APK应用程序

所有的 APK 应用程序要安装到 Android 仿真器上，使用 adb install 指令来开启一个命令字符的终端机窗口，并运行 APK 安装指令：

```
adb install filename.apk
```

这样 adb 指令就会自动将 filename.apk 应用程序安装到 Android 仿真器上，而仿真器上的应用程序列表也会立即出现刚刚安装的应用程序图标。如果应用程序没有安装成功，或安装不完善，也可以重复运行 adb install -r filename.apk 指令重新安装一次，这样会保留已经设置的信息，而仅是重新安装应用程序本身。

不过在运行 adb 安装 APK 应用程序组件时，不能同时运行多个 Android 仿真器，因为 adb 不知要将 APK 应用程序安装到哪一个仿真器，最好的方法就是仅运行一个 Android 仿真器。如果有同时运行多个仿真器的需要，就要在安装 APK 组件时，使用 adb 先指定某一个仿真器。可以从 Android 仿真器的窗口上，看到类似“Android Emulator(5554)”的字样，而 5554 就是仿真器的运行序号，每一个仿真器有其独特的运行序号，只要将 adb 加上-s <serialNumber> 参数，就可以指定 adb 将 APK 应用程序安装在哪一个仿真器上了：

```
adb -s emulator-5554 install filename.apk
(指定安装 APK 组件在 5554 的 Android 仿真器中)
```

3. 移除APK应用程序

如果已经安装了很多 Android 应用程序，若想删除一些应用程序图标也非常简单，同样是一行指令就搞定了。adb uninstall 指令可以将 APK 应用程序移除：

```
adb uninstall package
```



例如下面的代码：

```
adb uninstall com.android.email      (把 E-mail 程序移除)
```

Android 使用的 package 名称类似我们浏览网页时常用的域名方式，所以上面的示范是将 com.android.email 这个 email package 移除。请记住，package 名称不是您安装 APK 组件时的文件名或是显示在 Android 仿真器中的应用程序名称。另外 Package 名称也并不一定都是 com.android 这样的形式，它可以是各式各样的域名方式来命名，例如 org.iii.ro.iiivpa 或 com.deafcode.android.Cinema。APK 文件的 Package 名称完全是由当初的开发人员所制定的，所以并没有统一的命名方式，唯一相同的就是它一定是类似 Domain 域名的命名格式。

另外，在移除该 APK 应用程序时，如果想要保留信息与 Cache 目录，则加上 -k 参数即可。

```
adb uninstall -k package               (移除程序时，保留信息)
```

不过麻烦的是，可能不知道这个想要移除的应用程序的 Package 名称，所以必须先运行 adb shell 进入 Android 操作系统的指令列模式，然后到 /data/data 或 /data/app 目录下，得知欲移除的 Package 名称，然后使用 adb uninstall 指令删除 APK 应用程序，这样就可以简易地从 Android 仿真器中将不想使用的 APK 应用程序移除了。

```
adb shell
  ls /data/data 或 /data/app            (查询 Package 名称)
  exit
   adb uninstall package                (移除查询到的 Package)
```

幸运的是，从 Android SDK 1.5 版起，已经内建应用程序管理系统，不需要再辛苦地使用 adb uninstall 指令移除 APK 应用程序组件，只要在 Android 手机主界面点击 MENU 按键，然后依序点击“Settings→Applications→Manage applications”命令，就可以启动应用程序管理系统。当前 Android 系统已经安装的所有应用程序都会罗列出来，您只要点选想要移除的应用程序然后选择 uninstall 就可以移除该程序了，这样就不需要使用 adb uninstall 指令来移除 Android 应用程序了。

下载文件与打开网页是一样的，打开网页是将内容显示出来，保存文件就是保存到文件中。例如可以通过下面的代码将内容保存到 SD 卡等设备上。

```
public void downFile(String url, String path, String fileName)
          throws IOException {
       if (fileName == null || fileName == "")
          this.FileName = url.substring(url.lastIndexOf("/") + 1);
       else
          this.FileName = fileName; // 取得文件名，如果输入新文件名，则使用新文件名
       URL Url = new URL(url);
       URLConnection conn = Url.openConnection();
       conn.connect();
       InputStream is = conn.getInputStream();
       this.fileSize = conn.getContentLength();    // 根据响应获取文件大小
       if (this.fileSize <= 0) {                   // 获取内容长度为 0
          throw new RuntimeException("无法获知文件大小 ");
```

```
        }
        if (is == null) {        // 没有下载流
            sendMsg(Down ERROR);
            throw new RuntimeException("无法获取文件");
        }
        FileOutputStream FOS = new FileOutputStream(path + this.FileName);
           // 创建写入文件内存流，通过此流向目标写文件
        byte buf[] = new byte[1024];
        downLoadFilePosition = 0;
        int numread;
        while ((numread = is.read(buf)) != -1) {
            FOS.write(buf, 0, numread);
            downLoadFilePosition += numread
        }
        try {
            is.close();
        } catch (Exception ex) {
            ;
        }
    }
```

11.6.2　具体实现

本实例的功能是，能够远程下载指定网址的 Android 应用程序，下载到手机后打开 application installer 软件来安装这个软件。在具体实现上，先设置一个 EditText 来获取远程 URL，然后通过自定义按钮打开下载程序(使用 java.net 的 URLConnection 对象来创建连接，通过 InputStream 将下载文件写入到存储卡的缓存)，下载后通过自定义方法 openFile()打开文件，并根据文件扩展名，判断是否为 APK 格式，如果是则启动内置的 Install 程序，开始安装。安装完成后，在离开 Install 时通过方法 delFile()将存储卡中的临时文件删除。

实　例	功　能	源码路径
实例 11-4	开发一个远程下载系统	下载路径:\daima\11\xia

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下：

```
<TextView
  android:id="@+id/myTextView1"
  android:layout width="wrap content"
  android:layout height="wrap content"
  android:text="@string/str text"
>
</TextView>
<EditText
  android:id="@+id/myEditText1"
  android:layout width="fill parent"
  android:layout height="wrap content"
  android:text="@string/str_url"
```

```
    android:textSize="18sp"
  >
  </EditText>
  <Button
    android:id="@+id/myButton1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text="@string/str button"
  >
  </Button>
```

(2) 编写主程序文件 xia.java，其具体实现流程如下：

① 单击按钮后设置将文件下载到 local 本地，获取要安装程序的文件名称。具体代码如下：

```
mButton01.setOnClickListener(new Button.OnClickListener()
    {
      public void onClick(View v)
      {
        /* 文件会下载至 local 端 */
        mTextView01.setText("下载中...");
        strURL = mEditText01.getText().toString();
        /*取得欲安装程序的文件名称*/
        fileEx = strURL.substring(strURL.lastIndexOf(".")
        +1,strURL.length()).toLowerCase();
        fileNa = strURL.substring(strURL.lastIndexOf("/")
        +1,strURL.lastIndexOf("."));
        getFile(strURL);
       }
     }
    );
```

② 如果文本框中的远程地址为空，则输出“请输入 URL”的提示。具体代码如下：

```
    mEditText01.setOnClickListener(new EditText.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        // TODO Auto-generated method stub
        mEditText01.setText("");
        mTextView01.setText("远程安装程序(请输入 URL)");
      }
    });
  }
```

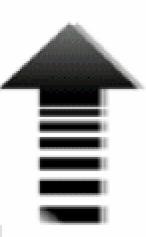
③ 定义方法 getFile(final String strPath)来获取下载的 URL 文件，如果有异常则输出提示。具体代码如下：

```
  /* 处理下载 URL 文件自定义函数 */
  private void getFile(final String strPath)  {
    try
```

```
  {
    if (strPath.equals(currentFilePath) )
    {
      getDataSource(strPath);
    }
    currentFilePath = strPath;
    Runnable r = new Runnable()
    {
      public void run()
      {
        try
        {
          getDataSource(strPath);
        }
        catch (Exception e)
        {
          Log.e(TAG, e.getMessage(), e);
        }
      }
    };
    new Thread(r).start();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
```

④ 定义方法 getDataSource 来获取远程文件，主要代码如下：

```
 /*取得远程文件*/
private void getDataSource(String strPath) throws Exception
{
  if (!URLUtil.isNetworkUrl(strPath))
  {
    mTextView01.setText("错误的 URL");
  }
  else
  {
    /*取得 URL*/
    URL myURL = new URL(strPath);
    /*创建连接*/
    URLConnection conn = myURL.openConnection();
    conn.connect();
    /*InputStream 下载文件*/
    InputStream is = conn.getInputStream();
    if (is == null)
    {
      throw new RuntimeException("stream is null");
    }
    /*创建临时文件*/
    File myTempFile = File.createTempFile(fileNa, "."+fileEx);
```




```
    /*取得暂存盘的路径*/
    currentTempFilePath = myTempFile.getAbsolutePath();
    /*将文件写入暂存盘*/
    FileOutputStream fos = new FileOutputStream(myTempFile);
    byte buf[] = new byte[128];
    do
    {
      int numread = is.read(buf);
      if (numread <= 0)
      {
        break;
      }
      fos.write(buf, 0, numread);
    }while (true);

    /*打开文件进行安装*/
    openFile(myTempFile);
    try
    {
      is.close();
    }
    catch (Exception ex)
    {
      Log.e(TAG, "error: " + ex.getMessage(), ex);
    }
  }
}
```

⑤ 定义方法 openFile(File f)来设置在手机上打开文件，主要代码如下：

```
/* 在手机上打开文件的 method */
private void openFile(File f)
{
  Intent intent = new Intent();
  intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
  intent.setAction(android.content.Intent.ACTION_VIEW);

  /* 调用 getMIMEType()来取得 MimeType */
  String type = getMIMEType(f);
  /* 设置 intent 的 file 与 MimeType */
  intent.setDataAndType(Uri.fromFile(f),type);
  startActivity(intent);
}
/* 判断文件 MimeType 的 method */
private String getMIMEType(File f)
{
  String type="";
  String fName=f.getName();
  /* 取得扩展名 */
  String end=fName.substring(fName.lastIndexOf(".")
  +1,fName.length()).toLowerCase();

  /* 依扩展名的类型决定 MimeType */
```

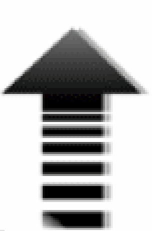
```
  if(end.equals("m4a")||end.equals("mp3")||end.equals("mid")||
  end.equals("xmf")||end.equals("ogg")||end.equals("wav"))
  {
    type = "audio";
  }
  else if(end.equals("3gp")||end.equals("mp4"))
  {
    type = "video";
  }
  else if(end.equals("jpg")||end.equals("gif")||end.equals("png")||
  end.equals("jpeg")||end.equals("bmp"))
  {
    type = "image";
  }
  else if(end.equals("apk"))
  {
    /* android.permission.INSTALL PACKAGES */
    type = "application/vnd.android.package-archive";
  }
  else
  {
    type="*";
  }
  /*如果无法直接打开，就跳出软件列表供用户选择 */
  if(end.equals("apk"))
  {
  }
  else
  {
    type += "/*";
  }
  return type;
}
```

⑥ 定义方法 delFile 来删除 SD 卡上的临时文件，主要代码如下：

```
/*自定义删除文件方法*/
private void delFile(String strFileName)
{
  File myFile = new File(strFileName);
  if(myFile.exists())
  {
    myFile.delete();
  }
}
```

⑦ 定义方法 onPause()和 onResume()，分别设置 onPause 暂停和 onResume 重新开始的状态。具体代码如下：

```
/*当 Activity 处于 onPause 状态时，更改 TextView 文字状态*/
@Override
protected void onPause()
```



```
  {
    mTextView01 = (TextView)findViewById(R.id.myTextView1);
    mTextView01.setText("下载成功");
    super.onPause();
  }
  /*当 Activity 处于 onResume 状态时，删除临时文件*/
  @Override
  protected void onResume()
  {
    // TODO Auto-generated method stub
    /* 删除临时文件 */
    delFile(currentTempFilePath);
    super.onResume();
  }
}
```

执行后将在文本框中显示目标安装程序的路径，如图 11-5 所示。实例中的默认路径是 http://mz.ruan8.com/soft/2/sougoushoujishurufa_7786.apk，这是一个 sogou 输入法程序。单击“安装”按钮后，开始下载目标文件，如图 11-6 所示。下载完成后弹出安装界面，单击 Install 按钮后开始安装，安装完成后输出提示。

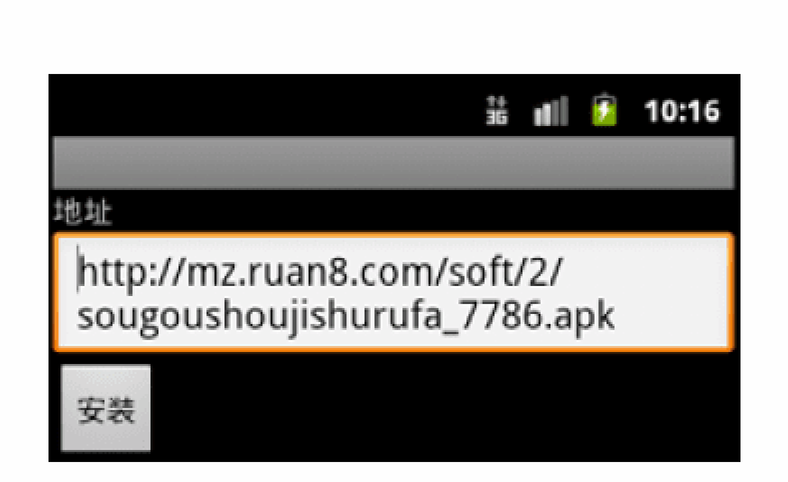


图 11-5　下载目标文件

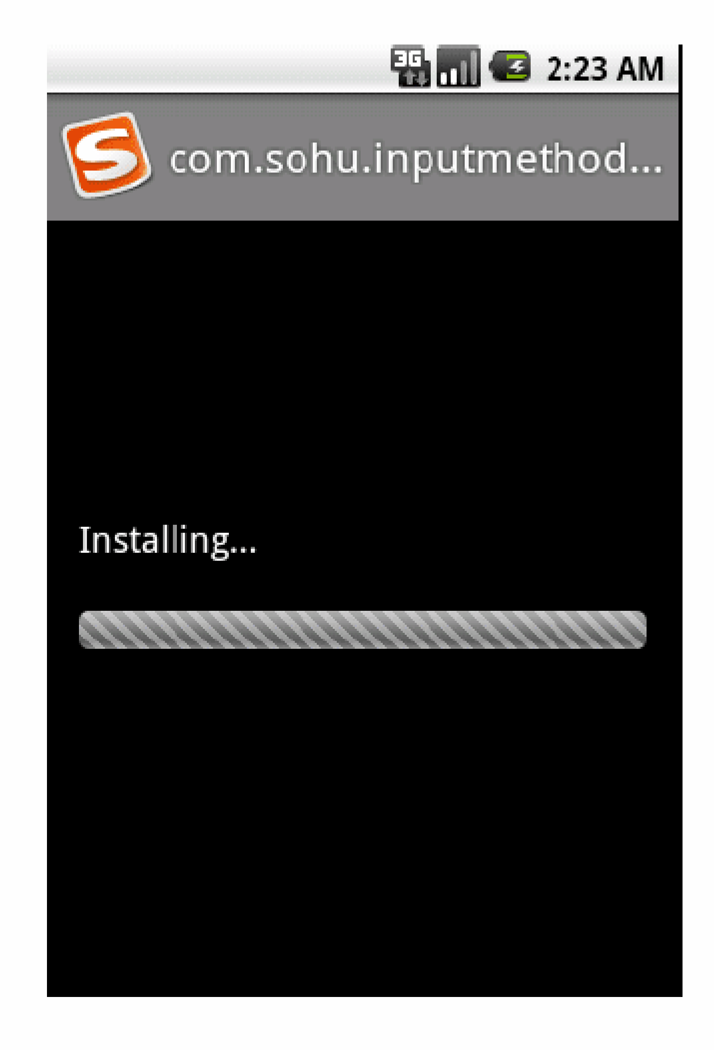


图 11-6　下载界面

11.7　在 Android 中开发一个网络视频播放器

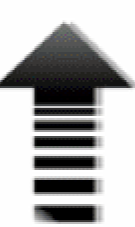
目前智能手机都可以远程观看在线视频，通常视频都比较大，所以必须保证手机空间能够存储，另外还要确保下载的视频能够被 MediaPlayer 所支持。在本实例中，通过 EditText 来获取远程视频的 URL，然后将此网址的视频下载到手机的存储卡中，以暂存的方式保存在 SD 卡中，再通过控制按钮来控制对视频的处理。在播放完毕并终止程序后，将暂存到 SD 卡中的临时视频删除。

实　例	功　能	源码路径
实例 11-5	使用 MediaPlayer 播放网络中的视频	下载路径:\daima\11\bof

11.7.1　实现布局文件

本实例的布局文件是 main.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:background="@drawable/white"
  android:orientation="vertical"
  android:layout width="fill parent"
  android:layout height="fill parent"
>
  <TextView
    android:id="@+id/myTextView1"
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:textColor="@drawable/blue"
    android:text="@string/hello" />
  <EditText
    android:id="@+id/myEditText1"
    android:layout width="wrap content"
    android:layout height="wrap content"
    android:text=""
    android:singleLine="True" />
  <SurfaceView
    android:id="@+id/mSurfaceView1"
    android:visibility="visible"
    android:layout width="320px"
    android:layout height="240px">
  </SurfaceView>
  <LinearLayout
    android:orientation="horizontal"
    android:layout height="wrap content"
    android:layout width="fill parent"
    android:padding="10dip"
  >
  <ImageButton android:id="@+id/play"
    android:layout height="wrap content"
    android:layout width="wrap content"
    android:src="@drawable/play"
  />
  <ImageButton android:id="@+id/pause"
    android:layout height="wrap content"
    android:layout width="wrap content"
    android:src="@drawable/pause"
  />
```



```
  <ImageButton android:id="@+id/reset"
    android:layout height="wrap content"
    android:layout width="wrap content"
    android:src="@drawable/reset"
  />
  <ImageButton android:id="@+id/stop"
    android:layout height="wrap content"
    android:layout width="wrap content"
    android:src="@drawable/stop"
  />
  </LinearLayout>
</LinearLayout>
```

11.7.2　实现显示文本值文件

本实例的屏幕显示文本值文件是 strings.xml，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello"></string>
  <string name="app name"></string>
  <string name="str_play">播放中</string>
  <string name="str_done">播放完毕</string>
  <string name="str_pause">暂停</string>
  <string name="str_stop">停止</string>
  <string name="str err nosd">※没有安装 SD 存储卡※</string>
</resources>
```

11.7.3　主程序文件

本实例的主程序文件是 bof.java，其具体实现流程如下。

(1) 定义 bIsReleased 来标识 MediaPlayer 是否已被释放，识别 MediaPlayer 是否正处于暂停，并用 LogCat 输出 TAG filter。具体代码如下：

```
  /* 识别 MediaPlayer 是否已被释放*/
  private boolean bIsReleased = false;
  /* 识别 MediaPlayer 是否正处于暂停*/
  private boolean bIsPaused = false;
  /* LogCat 输出 TAG filter */
  private static final String TAG = "HippoMediaPlayer";
  private String currentFilePath = "";
  private String currentTempFilePath = "";
  private String strVideoURL = "";
```

(2) 设置播放视频的 URL 地址，使用 mSurfaceView01 来绑定 Layout 上的 SurfaceView，然后设置 SurfaceHolder 为 Layout SurfaceView。具体代码如下：

```
  public void onCreate(Bundle savedInstanceState)
  {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
/* 将.3gp 图像文件存放 URL 网址*/
strVideoURL =
"http://new4.sz.3gp2.com//20100205xyy/喜羊羊与灰太狼%20 踩高跷
   (www.3gp2.com).3gp";
//http://www.dubblogs.cc:8751/Android/Test/Media/3gp/test2.3gp

mTextView01 = (TextView)findViewById(R.id.myTextView1);
mEditText01 = (EditText)findViewById(R.id.myEditText1);
mEditText01.setText(strVideoURL);

/* 绑定 Layout 上的 SurfaceView */
mSurfaceView01 = (SurfaceView) findViewById(R.id.mSurfaceView1);

/* 设置 PixnelFormat */
getWindow().setFormat(PixelFormat.TRANSPARENT);
/* 设置 SurfaceHolder 为 Layout  SurfaceView */
mSurfaceHolder01 = mSurfaceView01.getHolder();
mSurfaceHolder01.addCallback(this);
```

(3) 为影片设置大小比例，并分别设置 mPlay、mReset、mPause 和 mStop 四个控制按钮。具体代码如下：

```
/* 由于原有的影片 Size 较小，故指定其为固定比例*/
mSurfaceHolder01.setFixedSize(160, 128);
mSurfaceHolder01.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
mPlay = (ImageButton) findViewById(R.id.play);
mReset = (ImageButton) findViewById(R.id.reset);
mPause = (ImageButton) findViewById(R.id.pause);
mStop = (ImageButton) findViewById(R.id.stop);
```

(4) 编写单击“播放”按钮的处理事件，具体代码如下：

```
/* 播放按钮*/
mPlay.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    if(checkSDCard())
    {
      strVideoURL = mEditText01.getText().toString();
      playVideo(strVideoURL);
      mTextView01.setText(R.string.str_play);
    }
    else
    {
      mTextView01.setText(R.string.str_err_nosd);
    }
  }
});
```



(5) 编写单击“重播”按钮的处理事件，具体代码如下：

```
/* 重新播放按钮*/
mReset.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    if(checkSDCard())
    {
      if(bIsReleased == false)
      {
        if (mMediaPlayer01 != null)
        {
          mMediaPlayer01.seekTo(0);
          mTextView01.setText(R.string.str play);
        }
      }
    }
    else
    {
      mTextView01.setText(R.string.str err nosd);
    }
  }
});
```

(6) 编写单击“暂停”按钮的处理事件，具体代码如下：

```
/* 暂停按钮*/
mPause.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    if(checkSDCard())
    {
      if (mMediaPlayer01 != null)
      {
        if(bIsReleased == false)
        {
          if(bIsPaused==false)
          {
            mMediaPlayer01.pause();
            bIsPaused = true;
            mTextView01.setText(R.string.str pause);
          }
          else if(bIsPaused==true)
          {
            mMediaPlayer01.start();
            bIsPaused = false;
            mTextView01.setText(R.string.str play);
          }
        }
      }
```

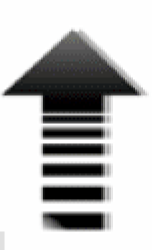
```
      }
      else
      {
        mTextView01.setText(R.string.str err nosd);
      }
    }
  });
```

(7) 编写单击“停止”按钮的处理事件，具体代码如下：

```
  /* 停止按钮*/
  mStop.setOnClickListener(new ImageButton.OnClickListener()
  {
    public void onClick(View view)
    {
      if(checkSDCard())
      {
        try
        {
          if (mMediaPlayer01 != null)
          {
            if(bIsReleased==false)
            {
              mMediaPlayer01.seekTo(0);
              mMediaPlayer01.pause();
              mTextView01.setText(R.string.str_stop);
            }
          }
        }
        catch(Exception e)
        {
          mTextView01.setText(e.toString());
          Log.e(TAG, e.toString());
          e.printStackTrace();
        }
      }
      else
      {
        mTextView01.setText(R.string.str err nosd);
      }
    }
  });
}
```

(8) 定义方法 playVideo 来下载指定 URL 地址的影片，并在下载后进行播放处理。具体代码如下：

```
/* 自定义下载 URL 影片并播放*/
private void playVideo(final String strPath)
{
  try
  {
```



```
/* 若传入的 strPath 为现有播放的连接，则直接播放*/
if (strPath.equals(currentFilePath) && mMediaPlayer01 != null)
{
  mMediaPlayer01.start();
  return;
}
else if(mMediaPlayer01 != null)
{
  mMediaPlayer01.stop();
}
currentFilePath = strPath;
/* 重新构建 MediaPlayer 对象*/
mMediaPlayer01 = new MediaPlayer();
/* 设置播放音量*/
mMediaPlayer01.setAudioStreamType(2);
/* 设置显示于 SurfaceHolder */
mMediaPlayer01.setDisplay(mSurfaceHolder01);
mMediaPlayer01.setOnErrorListener
(new MediaPlayer.OnErrorListener()
{
  @Override
  public boolean onError(MediaPlayer mp, int what, int extra)
  {
    // TODO Auto-generated method stub
    Log.i
    (
      TAG,
      "Error on Listener, what: " + what + "extra: " + extra
    );
    return false;
  }
});
```

(9) 定义 onBufferingUpdate 事件来监听缓冲进度，具体代码如下：

```
mMediaPlayer01.setOnBufferingUpdateListener
(new MediaPlayer.OnBufferingUpdateListener()
{
  @Override
  public void onBufferingUpdate(MediaPlayer mp, int percent)
  {
    // TODO Auto-generated method stub
    Log.i
    (
      TAG, "Update buffer: " +
      Integer.toString(percent) + "%"
    );
  }
});
```

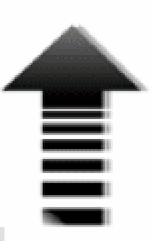
(10) 定义方法 run()来接收连接并记录线程信息。先在运行线程时调用自定义函数来抓

取文件，当下载完后调用 prepare 准备动作，当有异常发生时输出错误信息。其具体代码如下：

```
    Runnable r = new Runnable()
    {
      public void run()
      {
        try
        {
          /* 在线程运行中，调用自定义函数抓取文件*/
          setDataSource(strPath);
          /* 下载完后才会调用 prepare */
          mMediaPlayer01.prepare();
          Log.i
          (
            TAG, "Duration: " + mMediaPlayer01.getDuration()
          );
          mMediaPlayer01.start();
          bIsReleased = false;
        }
        catch (Exception e)
        {
          Log.e(TAG, e.getMessage(), e);
        }
      }
    };
    new Thread(r).start();
  }
  catch(Exception e)
  {
    if (mMediaPlayer01 != null)
    {
      mMediaPlayer01.stop();
      mMediaPlayer01.release();
    }
  }
}
```

(11) 定义方法 setDataSource 使用线程启动的方式来播放视频，具体代码如下：

```
/* 自定义 setDataSource，由线程启动*/
private void setDataSource(String strPath) throws Exception
{
  if (!URLUtil.isNetworkUrl(strPath))
  {
    mMediaPlayer01.setDataSource(strPath);
  }
  else
  {
    if(bIsReleased == false)
    {
```



```
        URL myURL = new URL(strPath);
        URLConnection conn = myURL.openConnection();
        conn.connect();
        InputStream is = conn.getInputStream();
        if (is == null)
        {
          throw new RuntimeException("stream is null");
        }
        File myFileTemp = File.createTempFile
        ("hippoplayertmp", "."+getFileExtension(strPath));

        currentTempFilePath = myFileTemp.getAbsolutePath();

        /*currentTempFilePath = /sdcard/mediaplayertmp39327.dat */

        FileOutputStream fos = new FileOutputStream(myFileTemp);
        byte buf[] = new byte[128];
        do
        {
          int numread = is.read(buf);
          if (numread <= 0)
          {
            break;
          }
          fos.write(buf, 0, numread);
        }while (true);
        mMediaPlayer01.setDataSource(currentTempFilePath);
        try
        {
          is.close();
        }
        catch (Exception ex)
        {
          Log.e(TAG, "error: " + ex.getMessage(), ex);
        }
      }
    }
  }
```

(12) 定义方法 getFileExtension 来获取视频的扩展名，具体代码如下：

```
  private String getFileExtension(String strFileName)
  {
    File myFile = new File(strFileName);
    String strFileExtension=myFile.getName();
    strFileExtension=(strFileExtension.substring
    (strFileExtension.lastIndexOf(".")+1)).toLowerCase();

    if(strFileExtension=="")
    {
      /* 若无法顺利取得扩展名，默认为.dat */
      strFileExtension = "dat";
```

```
    }
    return strFileExtension;
  }
```

(13) 定义方法 checkSDCard()来判断存储卡是否存在，具体代码如下：

```
  private boolean checkSDCard()
  {
    /* 判断存储卡是否存在*/
    if(android.os.Environment.getExternalStorageState().equals
    (android.os.Environment.MEDIA MOUNTED))
    {
      return true;
    }
    else
    {
      return false;
    }
  }
  @Override
  public void surfaceChanged
  (SurfaceHolder surfaceholder, int format, int w, int h)
  {
    Log.i(TAG, "Surface Changed");
  }
  public void surfaceCreated(SurfaceHolder surfaceholder)
  {
    Log.i(TAG, "Surface Changed");
  }

  @Override
  public void surfaceDestroyed(SurfaceHolder surfaceholder)
  {
    Log.i(TAG, "Surface Changed");
  }
}
```

执行后在文本框中显示指定播放视频的 URL，当下载完毕后能够实现播放处理，如图 11-7 所示。



图 11-7　执行效果



实例中的 MediaProvider 相当于一个数据中心，在里面记录了存储卡中的所有数据，而 Gallery 的作用就是展示和操作这个数据中心，每次用户启动 Gallery 时，Gallery 只是读取 MediaProvider 中的记录并显示用户。如果用户在 Gallery 中删除一个媒体时，Gallery 通过调用 MediaProvider 开放的接口来实现。

11.8　在 Android 中开发一个网络收音机

相信读者在自己的 Android 手机上见过网络收音机，这和传统的用天线来接收信号的收音机不同，网络收音机是通过网络来获取收音机信号的。本节将详细讲解在 Android 系统中开发网络收音机项目的基本知识。

11.8.1　基本思路

由于很多网络广播使用的协议是来自微软的 MMS，但是 Android 并不支持这种流媒体协议，读者可以尝试使用“Vitamio 插件+Vitamio 库”的方式来解决。这样在安装 app 本身的 APK 的同时还要安装对应你手机的 Vitamio 插件，这个插件是国外程序员开发的免费产品，支持很多媒体格式，大约 3MB。这个插件和硬件有关，读者在下载时可以一个一个地试。在笔者写作本书时，此插件已有如下 4 个版本。

- ARMv6: for some low end devices (Market, VOV)
- VFP: for some low end devices with VFP support (Market, VOV)
- ARMv7: for ARMv7 devices without NEON support, such as Tegra 2 powered devices(Market, VOV)
- NEON: for ARMv7 devices with NEON support (Market, VOV)

在具体开发之前，最好在 Vitamio 插件官网上下载 API 的使用文档 Vitamio-SDK.7z。在这个 SDK 里面还有一个名为 vitamio.jar 的 jar 文件，里面有流媒体的控制类。

11.8.2　演示代码

了解了开发 Android 网络收音机的思路，并准备好“Vitamio 插件+Vitamio 库”后，下面通过具体的演示代码来展示开发流程。

(1) AndroidManifest 文件的演示代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.netradiodemo"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="10" />
   <uses-permission android:name="android.permission.INTERNET"></uses-
       permission>

   <application android:icon="@drawable/icon" android:label=
```

```
        "@string/app name"
     android:theme="@android:style/Theme.Black">
        <activity android:name=".NetRadioDemoActivity"
                android:label="@string/app name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    <activity android:name=".compnents.PlayerActivity"></activity>
    </application>
</manifest>
```

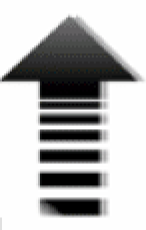
(2) UI 主页面布局文件 main 不用做任何修改，播放页面布局文件 play_page.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
<Button
    android:id="@+id/btn start"
    android:layout gravity="center horizontal"
    android:layout width="match parent"
    android:layout_height="wrap_content"
    android:text="听猫扑"
    android:textSize="30sp"
    android:onClick="doStart"
    />
    <Button
    android:id="@+id/btn stop"
    android:layout gravity="center horizontal"
    android:layout width="match parent"
    android:layout height="wrap content"
    android:text="不听猫扑"
    android:textSize="30sp"
    android:onClick="doStop"
    />
</LinearLayout>
```

(3) 实现第一个 Activity 代码，功能是检查插件 NetRadioDemoActivity，代码如下：

```
package com.netradiodemo;

import io.vov.vitamio.VitamioInstaller;
import io.vov.vitamio.VitamioInstaller.VitamioNotCompatibleException;
import io.vov.vitamio.VitamioInstaller.VitamioNotFoundException;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
```



```
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout.LayoutParams;
import android.widget.TextView;
import android.widget.Toast;
import com.netradiodemo.compnents.PlayerActivity;

public class NetRadioDemoActivity extends Activity {
     Intent intent ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        intent = new Intent(this, PlayerActivity.class);
        TextView tvCheck = new TextView(this);
        tvCheck.setText("使用前请检查是否安装了 Vitamio 插件:");
        this.addContentView(tvCheck, new LayoutParams(LayoutParams.MATCH_PARENT,
             LayoutParams.WRAP_CONTENT));

        Button btnCheck = new Button(this);
        btnCheck.setText("检查");
        btnCheck.setOnClickListener(new OnClickListener() {
              @Override
              public void onClick(View arg0) {
                  try {
                      String isInstallerString = VitamioInstaller
                          .checkVitamioInstallation(NetRadioDemoActivity.this);
     //检查插件是否安装成功，这里是一个费时操作，应该启用线程处理；作为一个 demo 我就不做了
                      Log.i("tag",isInstallerString);
                          //插件安装成功后，Log 中显示插件名称
                      if(isInstallerString!=null){
                          Toast.makeText(NetRadioDemoActivity.this, "已安装
                              正确版本 Vitamio!", Toast.LENGTH_LONG).show();
                          startActivity(intent);//开启收听界面
                      }else{
                          Toast.makeText(NetRadioDemoActivity.this, "没有匹配的
                              Vitamio!", Toast.LENGTH_LONG).show();
                         finish();//没有插件，安装失败，则结束程序
                      }
                  } catch (VitamioNotCompatibleException e) {
                      e.printStackTrace();
                  } catch (VitamioNotFoundException e) {
                      e.printStackTrace();
                  }
              }
         });
        this.addContentView(btnCheck, new LayoutParams(LayoutParams.WRAP_CONTENT,
             LayoutParams.WRAP_CONTENT));
    }
}
```

(4) 实现第二个 Activity，主要负责播放 PlayerActivity，代码如下：

```
package com.netradiodemo.compnents;

import io.vov.vitamio.MediaPlayer;
import io.vov.vitamio.VitamioInstaller.VitamioNotCompatibleException;
import io.vov.vitamio.VitamioInstaller.VitamioNotFoundException;
import io.vov.vitamio.widget.MediaController;
import java.io.IOException;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.LinearLayout.LayoutParams;

import com.netradiodemo.R;

public class PlayerActivity extends Activity {
    MediaPlayer mPlayer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        setContentView(R.layout.play page);
        MediaController controller = new MediaController(this);//创建控制对象
        this.addContentView(controller, new LayoutParams
            (LayoutParams.WRAP CONTENT, LayoutParams.WRAP CONTENT));
        String path = "mms://ting.mop.com/mopradio";
            //猫扑电台地址，这里可以添加自己的喜欢的电台地址，mms 协议的
        try {
            mPlayer = new MediaPlayer(this); //播放流媒体的对象
            mPlayer.setDataSource(path);      //设置流媒体的数据源
            mPlayer.prepare();
        } catch (VitamioNotCompatibleException e) {
            e.printStackTrace();
        } catch (VitamioNotFoundException e) {
            e.printStackTrace();
        } catch (IllegalArgumentException e) {
            e.printStackTrace();
        } catch (IllegalStateException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        super.onCreate(savedInstanceState);
    }

    public void doStart(View view){
        mPlayer.start();//开始播放
    }
    public void doStop(View view){
        mPlayer.stop();//停止播放
    }

}
```

通过上述编码操作之后，运行后就可以收听猫扑电台的音乐了。

Android

第 12 章

在Android中开发移动微博应用

微博，即微博客(MicroBlog)的简称，是一个基于用户关系的信息分享、传播以及获取平台，用户可以通过 Web、WAP 以及各种客户端组件个人社区，以 140 字左右的文字更新信息，并实现即时分享。本章将详细介绍在 Android 系统中开发微博项目的基本知识。

12.1 微博介绍

在互联网时代，使用博客的人越来越多，人们通过博客抒发情感、编写日记、记录生活中的点点滴滴，更有许多部落客，通过博客来分享不同领域的生活。为了方便人们的生活，在很多智能手机上推出了“移动博客发布器”。

最早也是最著名的微博是美国的 twitter，根据相关公开数据，截至 2010 年 1 月份，该产品在全球已经拥有 7500 万注册用户。2009 年 8 月份中国最大的门户网站新浪网推出“新浪微博”内测版，成为门户网站中第一家提供微博服务的网站，微博正式进入中文上网主流人群视野。

1. 微博的特点

微博客草根性更强，且广泛分布在桌面、浏览器、移动终端等多个平台上，有多种商业模式并存，或形成多个垂直细分领域的可能，但无论哪种商业模式，都离不开用户体验的特性和基本功能。

2. 手机微博

微博的主要发展运用平台应该是以手机用户为主，微博以电脑为服务器以手机为平台，把每个手机用户用无线的手机连在一起，让每个手机用户不用使用电脑就可以发表自己的最新信息，并和好友分享自己的快乐。

微博之所以要限定 140 个字符，就是源于从手机发短信最多字符 140 个(微博进入中国后普遍默认为 140 个汉字，随心微博 333 字)。可见微博从诞生之初就同手机应用密不可分，更是其在互联网形态中最大的亮点。微博对互联网的重大意义就在于建立手机和互联网应用的无缝连接，培养手机用户使用手机上网的习惯，增强手机端同互联网端的互动，从而使手机用户顺利过渡到无线互联网用户。目前手机和微博应用的结合有以下三种形式。

(1) 通过短信和彩信。

短信和彩信形式是同移动运营商合作，用户所花的短信和彩信费用由运营商收取，这种形式覆盖的人群比较广泛，只要能发短信就能更新微博，但对用户来说更新成本太大，并且彩信限制 50KB 的弊端严重影响了所发图片的清晰度。最关键的是这个方法只能提供更新，而无法看到其他人的更新，这种单向的信息传输方式大大降低了用户的参与性和互动性，让手机用户只体验到一个半吊子的微博。

(2) 通过 WAP 版网站。

各微博网站基本都有自己的 WAP 版，用户可以通过登录 WAP 或安装客户端连接到 WAP 版。这种形式只要手机能上网就能连接到微博，可以更新也可以浏览、回复和评论，所需费用就是浏览过程中使用的流量费。但目前国内的 GPRS 流量费还相对较高，网速也相对较慢，如果要上传大容量的图片，速度非常慢。

(3) 通过手机客户端。

手机客户端分以下两种。



① 微博网站开发的基于 WAP 的快捷方式版。

用户通过客户端直接连接到经过美化和优化的 WAP 版微博网站。这种形式用户行为主要靠主动来实现，也就是用户在想起更新和浏览微博的时候才打开客户端，其实也就相当于在手机端增加了一个微博网站快捷方式，使用操作上的利弊和 WAP 网站也基本相同。

② 利用微博网站提供的 API 开发的第三方客户端。

这种客户端在国内还比较少，国际上比较有名的是 twitter 的客户端 Gravity 和 Hesine(和信)。Gravity 是专门为 twitter 开发的，需要通过主动联网登录的，但操作架构和界面经过合理设计，用户体验非常好，可惜目前只支持 S60 系统。Hesine 是国内公司开发的，目前不但支持 twitter，还支持国内的各主流微博。与其他客户端不同的是，Hesine 的客户端是利用 IP Push 技术提供微博更新和下发通道，不但能够大大提升用户更新微博的速度，更重要的是能将微博消息推送到用户的手机，用户不用主动登录微博就能浏览和互动。Hesine 支持的系统平台比较多，但缺点是在非智能机上的体验还不够好。

相对于短信和彩信以及 WAP 形式，客户端的形式更符合无线互联网的发展趋势。尽管目前手机系统平台比较复杂，客户端开发起来难度很大，并且各客户端在非智能机上的发挥和体验整体都不佳，但是随着智能机逐渐平民化，无线网络速度的提升和流量资费的下调，手机和微博的结合肯定越来越密切，当山寨手机都能尽情地玩转微博的时候，相信那时候的微博会为互联网和 3G 应用带来很多革命性的变化。

12.2　微博开发技术介绍

本节将简单介绍在 Android 平台开发微博系统所需要的技术。

12.2.1　XML-RPC技术

开发移动微博的关键技术是 RPC。RPC 是 Remote Procedure Call 的缩写，意为“远程过程调用”。XML-RPC 是一种统一标准的规范，是通过 HTTP 连接的方式运行的，以传送符合 XML-RPC 格式的 Request 来调用远程服务器上的某个程序，进而运行博客功能。许多的网络服务业都会以 XML-RPC 方式提供给软件开发者一个系统连接的管道，让开发者能够根据业者定义好的方式，以 XML-RPC 的方式来使用该网站的某些功能。目前许多的博客也都支持 XML-RPC 的连接方式。

XML-RPC 的原理是，XML-RPC 工具把传入的参数组合成 XML，然后通过 HTTP 协议发送给服务器，服务器回复 XML 格式数据，再由工具解析给调用者。

在 XML-RPC 标准中，规定 XML 内容的规则如下：

```
<xml version=”1.0”?>
<methodCall>
  <methodName>要调用的 method name</methodName>
    <params>
    <params>参数 1</param>
    <params>参数 2</param>
```

```
        <param>参数 n</param>
    </params>
</methodCall>
```

Android 本身并不支持 XML-RPC 协议，需要下载相应的工具。读者可以从如下地址下载 XML-RPC：

```
http://code.google.com/p/android-xmlrpc/downloads/list
```

例如下面的代码演示了用 XML-RPC 协议实现微博客户端的基本过程。

```
package org.xmlrpc;

import java.net.URI;
import java.util.HashMap;
import java.util.Map;
import org.apache.http.conn.HttpHostConnectException;
import org.xmlrpc.android.XMLRPCClient;
import org.xmlrpc.android.XMLRPCException;
import org.xmlrpc.android.XMLRPCFault;
import org.xmlrpc.android.XMLRPCSerializable;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.util.Log;
import android.widget.EditText;
import android.widget.Toast;
import android.widget.Button;
import android.content.DialogInterface.OnCancelListener;
import android.view.View.OnClickListener;
import android.view.View;

public class TestBlog extends Activity {
    private XMLRPCClient client;
    private URI uri;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.test blog);
        Button btn = (Button) findViewById(R.id.send);
        btn.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                post();
            }
        });
    }

    void post() {
        String blogid = ((EditText) findViewById(R.id.blogid edit)).getText()
            .toString();        //ID 号
```

```
                String username = ((EditText) findViewById(R.id.username_edit))
                       .getText().toString();   //用户名
                String password = ((EditText) findViewById(R.id.password_edit))
                        .getText().toString();  // 密码
              String title = ((EditText) findViewById(R.id.title_edit)).getText()
                         .toString();           //标题
                String content = ((EditText) findViewById(R.id.content_edit))
                         .getText()
                         .toString();           // 正文
              uri = URI.create("http://blog.csdn.net/" + blogid
                   + "/services/metablogapi.aspx");
                client = new XMLRPCClient(uri);

                Map<String, Object> structx = new HashMap<String, Object>();
                structx.put("title", title);
                structx.put("description", content);
                Object[] params = new Object[] { blogid, username, password,
                     structx, true };

                try {
                          client.callEx("metaWeblog.newPost", params);
                          Toast.makeText(this, "OK", 10000).show();
                               } catch (XMLRPCException e) {
                          Toast.makeText(this, "ERROR" + e, 10000).show();
                }
        }
}
```

12.2.2　Meta Weblog API客户端

Meta Weblog API 是博客园发布的一款功能强大的客户端，其登录地址是 http://www.cnblogs.com/<您的用户名>/services/metaweblog.aspx。Meta Weblog API 支持通过 XML-RPC 的方法在软件中编辑及浏览 Blog，其常用的 API 如下。

- 发布新文章(metaWeblog.newPost)
- 获取分类(metaWeblog.getCategories)
- 最新文章(metaWeblog.getRecentPosts)
- 新建文章分类(wp.newCategory)
- 上传图片音频或视频(metaWeblog.newMediaObject)

12.3　在 Android 上开发移动博客发布器

在本实例中实现了一个“移动博客发布器”的功能，以乐多博客为例，演示了如何从手机发布文章到乐多博客上的方法。

实　例	功　能	源码路径
实例 12-1	开发一个移动微博发布系统	下载路径:\daima\12\weib

12.3.1　XML请求

调用乐多博客的 metaWeblog.newPost 接口实现添加博客文章的功能，发出的 XML 请求内容如下：

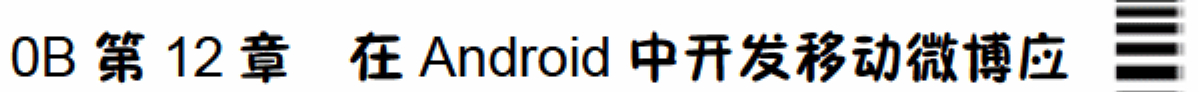
```
< ?xml version="1.0"?>
<methodCall>
      <methodN ame >metaWeblog.newPost</methodName>
<params>
             <param><value><st ring>ID</string></value></param>
             <param><value><string>账号</string></value></param>
             <param><value><string>密码</string></value></param>
<param>
 <value>
 <struct>
 <member>
<name>title</name>
        <value><string>文章标题</string></value>
 </member>
 <member>
<name>descriptiori</name>
        <value><string>内容</string></value>
 </member>
 </struct>
 </value>
</param>
<param><value><boolean>1</boolean></value></param>
</params>
</methodCall>
```

12.3.2　常用接口

并非所有的博客都可以用 Get 或 Post 方式实现 XML-RPC 的 Request 交互，有些博客只能以 Post 的方式来传送。所以在具体编码之前，要先弄清楚服务器接收 Request 是否有特殊限制。在乐多博客的项目中，为开发人员提供了许多交互方法，通过这些方法可以实现包罗万象的功能，在表 12-1 中列出了几种常用的方法。

表 12-1　常用的方法接口

方法名称	参　数	返 回 值	说　明
metaWeblog.newPost	博客 ID(string) usemame(string) password(string) content publish(boolean)	成功：文章 ID 失败：fault	发布一篇新文章




续表

方法名称	参　数	返 回 值	说　明
metaWeblog.editPost	文章 ID(string) usemame(string) password(string) content publish(boolean)	成功：true 失败：fault	修改已发布的文章内容
metaWeblog.getPost	文章 ID(string) usemame(string) password(string)	成功：文章数组 失败：fault	取得特定文章的信息
metaWeblog. getRecentPosts	博客 ID(string) usemame(string) password(string) 返回篇数(int)	成功：文章数组 失败：fault	返回最近发表的文章信息
metaWeblog.deletePost	文章 ID(string) usemame(string) password(string)	成功：true 失败：fault	删除已发布的博客文章
mt.getCategoryList	博客 ID(string) usemame(string) password(string)	成功：分类数组 失败：fault	取得博客的文章分类信息
mt.getPostCategories	文章 ID(string) usemame(string) password(string)	成功：分类数组 失败：fault	返回指定文章的所属类信息
mt.setPostCategories	文章 ID(string) usemame(string) password(string) 文章分类(数组)	成功：true 失败：fault	设置指定文章所在的类
mt.supportedMethods		成功：方法数组	取得服务器支持的 XML-RPC 方法列表

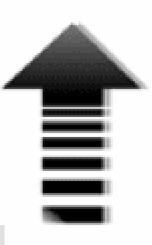
12.3.3　具体实现

在本实例中，以 EditText 编辑框作为输入博客相关信息及文章内容的组件，当用户输入完成后单击“发布文章”按钮，此 Button 按钮的 onClick()会被触发，首先检查输入字段是否为空白，检查无误后，程序先运行 getPostString()，将输入参数转换成符合 XML-RPC 规范的 XML 格式，再调用 sendPost()将 XML 的 Request 传送给相对应的博客网址，最后再取得服务器返回的 Response，并使用对话框形式显示运行结果。

本实例的具体实现流程如下。

(1) 编写布局文件 main.xml，主要代码如下：

```
<TextView
  android:id="@+id/myText1"
  android:layout_width="wrap_content"
  android:layout_height="23px"
  android:text="@string/str_title1"
  android:textColor="@drawable/black"
  android:layout_x="10px"
  android:layout_y="22px"
>
</TextView>
<TextView
  android:id="@+id/myText2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/str_title2"
  android:textColor="@drawable/black"
  android:layout_x="10px"
  android:layout_y="62px"
>
</TextView>
<TextView
  android:id="@+id/myText3"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/str_title3"
  android:textColor="@drawable/black"
  android:layout_x="10px"
  android:layout_y="102px"
>
</TextView>
<TextView
  android:id="@+id/myText4"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/str_title4"
  android:textColor="@drawable/black"
  android:layout_x="10px"
  android:layout_y="142px"
>
</TextView>
<TextView
  android:id="@+id/myText5"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/str_title5"
  android:textColor="@drawable/black"
  android:layout_x="10px"
  android:layout_y="182px"
>
```



```
</TextView>
<EditText
  android:id="@+id/blogId"
  android:layout_width="100px"
  android:layout_height="40px"
  android:numeric="integer"
  android:layout_x="90px"
  android:layout_y="12px"
>
</EditText>
<EditText
  android:id="@+id/blogAccount"
  android:layout_width="170px"
  android:layout_height="40px"
  android:textSize="16sp"
  android:layout_x="90px"
  android:layout_y="52px"
>
</EditText>
<EditText
  android:id="@+id/blogPwd"
  android:layout_width="170px"
  android:layout_height="40px"
  android:textSize="16sp"
  android:password="true"
  android:layout_x="90px"
  android:layout_y="92px"
>
</EditText>
<EditText
  android:id="@+id/artContent"
  android:layout_width="210px"
  android:layout_height="207px"
  android:textSize="16sp"
  android:layout_x="90px"
  android:layout_y="172px"
>
</EditText>
<EditText
  android:id="@+id/artTitle"
  android:layout_width="200px"
  android:layout_height="40px"
  android:textSize="16sp"
  android:layout_x="90px"
  android:layout_y="132px"
  android:scrollbars="vertical"
>
</EditText>
<Button
  android:id="@+id/myButton"
  android:layout_width="90px"
```

```
    android:layout height="40px"
    android:text="@string/str button"
    android:textSize="16sp"
    android:layout x="120px"
    android:layout_y="382px"
  >
  </Button>
```

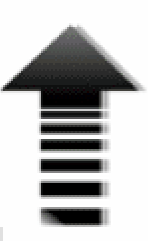
(2) 编写界面显示文本文件 strings.xml，主要代码如下：

```
<resources>
  <string name="hello"></string>
  <string name="app name"></string>
  <string name="str_title1">ID 号是：</string>
  <string name="str title2">登录账号：</string>
  <string name="str title3">登录密码：</string>
  <string name="str title4">文章标题：</string>
  <string name="str title5">文章内容：</string>
  <string name="str button">发布文章</string>
</resources>
```

(3) 编写主程序文件 weib.java，主要代码如下：

```
public class weib extends Activity
{
  /* 变量声明 */
  Button mButton;
  EditText mEdit1;
  EditText mEdit2;
  EditText mEdit3;
  EditText mEdit4;
  EditText mEdit5;
  /* 乐多博客 XML-RPC 网址 */
  private String path=
    " http://blog.csdn.net/asdfg343442";
  /* XML-RPC 发布文章的 method name */
  private String method="metaWeblog.newPost";

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /* 初始化对象 */
    mEdit1=(EditText)findViewById(R.id.blogId);
    mEdit2=(EditText)findViewById(R.id.blogAccount);
    mEdit3=(EditText)findViewById(R.id.blogPwd);
    mEdit4=(EditText)findViewById(R.id.artTitle);
    mEdit5=(EditText)findViewById(R.id.artContent);
    mButton=(Button)findViewById(R.id.myButton);
    /* 设置发布文章的 onClick 事件 */
    mButton.setOnClickListener(new View.OnClickListener()
```



```
    {
      public void onClick(View v)
      {
        /* 取得输入的信息 */
        String blogId=mEdit1.getText().toString();
        String account=mEdit2.getText().toString();
        String pwd=mEdit3.getText().toString();
        String title=mEdit4.getText().toString();
        String content=mEdit5.getText().toString();

        if(blogId.equals("")||account.equals("")||pwd.equals("")||
           title.equals("")||content.equals(""))
        {
          showDialog("没有填写内容!");
        }
        else
        {
          /* 发送 XML Post 并显示 Response 内容 */
          String outS=getPostString(method,blogId,account,
                   pwd,title,content);
          String re=sendPost(outS);
          showDialog(re);
        }
      }
    });
  }

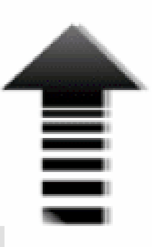
  /* 发送 Request 至博客的对应网址的 method */
  private String sendPost(String outString)
  {
    HttpURLConnection conn=null;
    String result="";
    URL url = null;
    try
    {
      url = new URL(path);
      conn = (HttpURLConnection)url.openConnection();
      /* 允许 Input、Output */
      conn.setDoInput(true);
      conn.setDoOutput(true);
      /* 设置传送的 method=POST */
      conn.setRequestMethod("POST");
      /* setRequestProperty */
      conn.setRequestProperty("Content-Type", "text/xml");
      conn.setRequestProperty("Charset", "UTF-8");

      /* 送出 Request */
      OutputStreamWriter out =
        new OutputStreamWriter(conn.getOutputStream(), "utf-8");
      out.write(outString);
      out.flush();
```




```
      out.close();
      /* 解析返回的 XML 内容 */
      result=parseXML(conn.getInputStream());
      conn.disconnect();
    }
    catch(Exception e)
    {
      conn.disconnect();
      e.printStackTrace();
      showDialog(""+e);
    }
    return result;
  }

  /* 解析 Response 的 XML 内容的 method */
  private String parseXML(InputStream is)
  {
    String result="";
    Document doc = null;
    try
    {
      /* 将 XML 转换成 Document 对象 */
      DocumentBuilderFactory dbf=
        DocumentBuilderFactory.newInstance();
      DocumentBuilder db=dbf.newDocumentBuilder();
      doc = db.parse(is);
      doc.getDocumentElement().normalize();
      /* 检查返回值是否有包含 fault 这个 tag，如果有就代表发布错误 */
      int fault=doc.getElementsByTagName("fault").getLength();
      if(fault>0)
      {
        result+="发布错误!\n";
        /* 取得 faultCode(错误代码) */
        NodeList nList1=doc.getElementsByTagName("int");
        for (int i = 0; i < nList1.getLength(); ++i)
        {
          String errCode=nList1.item(i).getChildNodes().item(0)
               .getNodeValue();
          result+="错误代码: "+errCode+"\n";
        }
        /* 取得 faultString(错误信息) */
        NodeList nList2=doc.getElementsByTagName("string");
        for (int i = 0; i < nList2.getLength(); ++i)
        {
          String errString=nList2.item(i).getChildNodes().item(0)
               .getNodeValue();
          result+="错误信息: "+errString+"\n";
        }
      }
      else
      {
```



```
      /* 发布成功，取得文章编号 */
      NodeList nList=doc.getElementsByTagName("string");
      for (int i = 0; i < nList.getLength(); ++i)
      {
        String artId=nList.item(i).getChildNodes().item(0)
            .getNodeValue();
        result+="发布成功!!文章编号「"+artId+"」";
      }
    }
  }
  catch (Exception ioe)
  {
    showDialog(""+ioe);
  }
  return result;
}

/* 一组要发送的 XML 内容的 method */
private String getPostString(String method,String blogId,
    String account,String pwd,String title,String content)
{
  String s="";
  s+="<methodCall>";
  s+="<methodName>"+method+"</methodName>";
  s+="<params>";
  s+="<param><value><string>"+blogId+"</string></value></param>";
  s+="<param><value><string>"+account+"</string></value></param>";
  s+="<param><value><string>"+pwd+"</string></value></param>";
  s+="<param><value><struct>";
  s+="<member><name>title</name>" +
     "<value><string>"+title+"</string></value></member>";
  s+="<member><name>description</name>" +
     "<value><string>"+content+"</string></value></member>";
  s+="</struct></value></param>";
  s+="<param><value><boolean>1</boolean></value></param>";
  s+="</params>";
  s+="</methodCall>";

  return s;
}

/* 跳出 Dialog 的 method */
private void showDialog(String mess)
{
  new AlertDialog.Builder(weib.this).setTitle("Message")
  .setMessage(mess)
  .setNegativeButton("确定", new DialogInterface.OnClickListener()
  {
    public void onClick(DialogInterface dialog, int which)
    {
    }
```




```
      })
      .show();
    }
}
```

执行后的效果如图 12-1 所示，只要拥有乐多的账号，就可以在手机上发送移动博客。



图 12-1　执行效果

12.4　详解腾讯 Android 版微博 API

作为一名 Android 学习者来说，个人独立开发微博系统的难度比较大。在目前市面中有很多著名微博系统的开源代码，开发人员只需利用它们提供的 API 接口即可方便地开发出 Android 版的移动微博系统。当今市面中著名的 Android 版的移动微博系统有新浪微博和腾讯微博。本节将讲解腾讯微博系统的 API 接口。

12.4.1　源码和jar包下载

因为当前腾讯微博提供的 Java(Android) SDK 功能过弱，所以特意集成了一个 Java SDK 包，此包适用于 Android。Java SDK 包含了腾讯微博目前提供的 95%的 API，用法简单(微博、评论、转发、私信同一个实体类)，方便扩展(可以根据自己的需要修改源代码或是继承 QqTSdkService 类，当然为了后续依然能升级版本，建议采用继承的方式)。

在压缩包中，QqTAndroidSdk-1.0.0.jar 是 SDK 的主代码，其中 QqTSdkServiceImpl 包含了所有接口的实现，具体说明如下。

- jar 包地址：QqTAndroidSdk-1.0.0.jar。
- google code 源码地址：http://code.google.com/p/qq-t-java-sdk/source/browse/。
- github 源码地址：https://github.com/Trinea/qq-t-java-sdk。

而压缩包 JavaCommon-1.0.0.jar 是 QqTAndroidSdk 依赖的公用处理包，包含了字符串、list、数组、map、json 工具类等，具体说明如下。

- jar 包地址：已经包含在 QqTAndroidSdk-1.0.0.jar 中。
- google code 源码地址：http://code.google.com/p/trinea-java-common/source/browse/。
- github 源码地址：https://github.com/Trinea/JavaCommon。

12.4.2　具体使用

在具体使用之前，读者请先参考腾讯微博的 API 文档说明，地址是：http://wiki.open.t.qq.com/index.php/API%E6%96%87%E6%A1%A3，如图 12-2 所示。



图 12-2　腾讯微博的API文档页面

在编码使用 API 接口时，需要先新建 QqTSdkService 类对象并进行初始化工作。例如下面的初始化代码：

```
/**
 * 分别设置应用的 key、secret(腾讯提供)。用户的 accesstoken 和 tokenSecret(OAuth 获取)
 * 请用自己的相应字符串替换，否则无法成功发送和获取数据
 **/
QqTAppAndToken qqTAppAndToken = new QqTAppAndToken();
qqTAppAndToken.setAppKey("***");         // ***用应用 key 替换
qqTAppAndToken.setAppSecret("***");      // ***用应用 secret 替换
qqTAppAndToken.setAccessToken("***");    // ***用用户 accesstoken 替换
qqTAppAndToken.setTokenSecret("***");    // ***用用户 tokenSecret 替换

/** 新建 QqTSdkService 对象，并设置应用信息和用户访问信息 **/
QqTSdkService qqTSdkService = new QqTSdkServiceImpl();
qqTSdkService.setQqTAppAndToken(qqTAppAndToken);
```

接下来开始对接口进行详细介绍，并举例如何使用 QqTAndroidSdk-1.0.0.jar 中的 API。腾讯微博中的接口主要分成下面的几大类。

1. 时间线(微博列表)

这里的 20 个接口包含了腾讯微博四部分 API。

(1) 时间线中的除 statuses/ht_timeline_ext(话题时间线)以外的 15 个 API。

(2) 私信相关中的收件箱、发件箱两个 API。

(3) 数据收藏中的收藏的微博列表和获取已订阅话题列表两个 API。

(4) 微博相关中的获取单条微博的转发或点评列表 API。

以获取首页信息为例，示例代码如下：

```
QqTTimelinePara qqTTimelinePara = new QqTTimelinePara();
/** 设置分页标识 **/
qqTTimelinePara.setPageFlag(0);
/** 设置起始时间 **/
qqTTimelinePara.setPageTime(0);
/** 每次请求记录的条数 **/
qqTTimelinePara.setPageReqNum(QqTConstant.VALUE_PAGE_REQ_NUM);
/** 可以设置拉取类型，可取值 QqTConstant 中 VALUE_STATUS_TYPE_TL … **/
qqTTimelinePara.setStatusType(QqTConstant.VALUE_STATUS_TYPE_TL_ALL);
/** 可以设置微博内容类型，可取值 QqTConstant 中 VALUE_CONTENT_TYPE_TL… **/
qqTTimelinePara.setContentType(QqTConstant.VALUE_CONTENT_TYPE_TL_ALL);
List<QqTStatus> qqTStatusList = qqTSdkService.getHomeTL(qqTTimelinePara);
assertTrue(qqTStatusList != null);
```

这样 qqTStatusList 就保存了首页的 20 条数据，可以自行设置不同的类型参数。如果想获取更多的时间线数据，请读者参考腾讯微博 Java(Android) SDK 时间线 API 的详细介绍。

2. 新增微博API

在本书成稿时，腾讯微博新增加了 8 个 API。

(1) 微博相关中的发表一条微博、转播一条微博、回复一条微博、发表一条带图片微博、点评一条微博、发表音乐微博、发表视频微博、发表心情帖子。在 API 中发表一条微博和发表一条带图片微博合二为一。

(2) 私信相关中的发私信操作一条微博。

以新增一条微博为例，示例代码如下：

```
qqTSdkService.addStatus("第一条状态哦", null);
```

其中第一个参数为状态内容，第二个参数为图片地址，不传图片为空即可。或者在如下复杂代码中，status 可以设置其他地理位置信息。

```
QqTStatusInfoPara status = new QqTStatusInfoPara();
status.setStatusContent("发表一条带图片微博啦");
/** 发表带图微博，设置图片路径 **/
status.setImageFilePath("/mnt/sdcard/DCIM/Camera/IMAG2150.jpg");
assertTrue(qqTSdkService.addStatus(status, qqTAppAndToken));
```



这 8 个接口包含了腾讯微博两部分 API。

3. 操作一条微博

(1) 微博相关中的删除一条微博 API。

(2) 私信相关中的删除私信 API。

(3) 数据收藏中的收藏微博、取消收藏微博、订阅话题、取消订阅话题 4 个 API。

以收藏一条微博为例，示例代码如下：

```
qqTSdkService.collect(12121);
```

其中参数为微博 ID。

4. 关系链列表(用户列表)

这 10 个接口包含了腾讯微博关系链相关中的互听关系链列表(对某个用户而言，既是他的听众又被他收听)、其他账号听众列表、其他账号收听的人列表、其他账户特别收听的人列表、黑名单列表、我的听众列表、我的听众列表(只包含名字)、我收听的人列表、我收听的人列表(只包含名字)、我的特别收听列表 10 个 API。

以获取自己的收听用户为例，示例代码如下所示：

```
QqTUserRelationPara qqTUserRelationPara = new QqTUserRelationPara();
qqTUserRelationPara.setReqNumber(QqTConstant.VALUE PAGE REQ NUM);
qqTUserRelationPara.setStartIndex(0);
List<QqTUser> qqTUserList =
qqTSdkService.getSelfInterested(qqTUserRelationPara);
```

5. 用户建立关系

这 6 个接口包含了腾讯微博关系链相关中的收听某个用户、取消收听某个用户、特别收听某个用户、取消特别收听某个用户、添加某个用户到黑名单、从黑名单中删除某个用户共 6 个 API。

以关注某些用户为例，示例代码如下：

```
qqTSdkService.interestedInOther("wenzhang,li_nian,mayili007", null)
```

6. 账户相关

这 7 个接口包含了腾讯微博账户相关中的获取自己的详细资料、更新用户信息、更新用户头像信息、更新用户教育信息、获取其他人资料、获取一批人的简单资料、验证账户是否合法(是否注册微博)共 7 个 API。除获取心情微博 API 外。

以获取自己的资料为例，示例代码如下：

```
QqTUser qqTUser = qqTSdkService.getSelfInfo();
```

7. 搜索相关

这 3 个接口包含了腾讯微博搜索相关中的搜索用户、搜索微博、通过标签搜索用户共 3 个 API。以搜索微博为例，示例代码如下：

```
public void testSearchStatus() {
    QqTSearchPara qqTSearchPara = new QqTSearchPara();
    qqTSearchPara.setKeyword("iphone");
    qqTSearchPara.setPage(1);
    qqTSearchPara.setPageSize(QqTConstant.VALUE PAGE REQ NUM);
    List<QqTStatus> qqTStatusList =
qqTSdkService.searchStatus(qqTSearchPara);
    assertTrue(qqTStatusList != null);
}
```

8. 热度趋势相关

这两个接口包含了腾讯微博热度趋势中的话题热榜、转播热榜用户共两个 API。以话题热榜为例，示例代码如下：

```
public void testGetHotTopics() {
    QqTHotStatusPara qqTHotStatusPara = new QqTHotStatusPara();
    qqTHotStatusPara.setReqNum(QqTConstant.VALUE PAGE REQ NUM);
    qqTHotStatusPara.setLastPosition(0);
    /**
     * 1：话题名；2：搜索关键字；3：两种类型都有
     **/
    qqTHotStatusPara.setType(Integer.toString(1));
    List<QqTTopicSimple> hotTopicsList =
qqTSdkService.getHotTopics(qqTHotStatusPara);
    assertTrue(hotTopicsList != null);
}
```

9. 数据更新

这一个接口为腾讯微博数据更新相关中的查看数据更新条数 API，示例代码如下：

```
public void testGetUpdateInfoNum() {
    /** 设置 clearType，对应 QqTConstant.VALUE_CLEAR_TYPE_… **/
    QqTUpdateNumInfo qqTUpdateNumInfo =
qqTSdkService.getUpdateInfoNum(true,
QqTConstant.VALUE CLEAR TYPE HOME PAGE);
    assertTrue(qqTUpdateNumInfo != null);
}
```

10. 发起话题

这两个接口是为腾讯微博中的话题应用服务的，可以根据话题名称查询话题 ID 和根据话题 ID 获取话题相关信息 API。示例代码如下：

```
public void testGetTopicInfoByIds() {
    /** 先得到话题 ID **/
    Map<String, String> topicIdAndName =
qqTSdkService.getTopicIdByNames("袁莉闪婚,美汁源下架,iphone");
    if (topicIdAndName != null) {
```



```
        /** 话题 ID 列表，以逗号分隔 **/
        List<QqTStatus> qqtStatusList = qqTSdkService.getTopicInfoByIds
          (ListUtils.join(new ArrayList<String>(topicIdAndName.keySet())));
        assertTrue(qqtStatusList != null);
    } else {
        assertTrue(false);
    }
}
```

11. 标签相关

这两个接口为腾讯微博标签相关中的添加标签和删除标签 API，示例代码如下：

```
public void testDeleteTag() {
    /** 删除自己的 tag，先获取自己的资料，从中取中 tag id **/
    QqTUser qqTUser = qqTSdkService.getSelfInfo();
    if (qqTUser != null && qqTUser.getTagMap() != null &&
         qqTUser.getTagMap().size() > 0) {
        /** 删除 tag **/
        for (Map.Entry<String, String> tag :
              qqTUser.getTagMap().entrySet()) {
            qqTSdkService.deleteTag(tag.getKey());
        }
    } else {
        assertTrue(false);
    }
}
```

12.5　详解新浪 Android 版微博 API

新浪微博是国内最早推出微博应用的行业站点，为了帮助 Android 程序员开发可以使用新浪微博的应用，特意提供了开源 API 供大家参考。读者要想了解在 Android 平台使用新浪微博的知识，可以登录 http://open.weibo.com/wiki/%E9%A6%96%E9%A1%B5 获取详细资料，并且在网页中可以获取开源代码。

在 Android 中使用新浪微博的开发平台 API 的基本步骤如下。

1. 通过官方网址下载SDK

当前的最新版本是 Weibo4Android，下载地址是：

http://code.google.com/p/weibo4j/downloads/detail?name=weibo4android-1.2.1.zip

此页面的界面效果如图 12-3 所示。



图 12-3　下载SDK页面

2. 认证

在 SDK 中有完整的如何通过 OAuth 认证的演示实例。认证和使用流程如下。

(1) 在/weibo4android/src/weibo4android/Weibo.java 设置 App Key 和 App Secret(在官方网站新建应用可获得)，如下所示：

```
public static String CONSUMER_KEY = "2664209963";
public static String CONSUMER_SECRET =
"b428615797a5d676d428cd146c040399";
```

(2) 在/weibo4android/examples/weibo4android/androidexamples/AndroidExample.java 中，将 App Key 和 App Secret 设置进系统类中：

```
System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
System.setProperty("weibo4j.oauth.consumerSecret",
Weibo.CONSUMER_SECRET);
```

(3) 通过 HTTP Post 方式向服务提供方请求获得 RequestToken。

```
RequestToken  requestToken
=weibo.getOAuthRequestToken("weibo4android://OAuthActivity");
```

(4) 将用户引导至授权页面。

```
Uri uri = Uri.parse(requestToken.getAuthenticationURL()+
"&display=mobile");
startActivity(new Intent(Intent.ACTION_VIEW, uri));
```

(5) 授权页面要求用户输入用户名和密码，授权完成后，服务提供方会通过回调 URL



将用户引导回客户端页面 OAuthActivity 页面。

```
    <activity android:name=".OAuthActivity">
     <intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <category android:name="android.intent.category.BROWSABLE" />
      <data android:scheme="weibo4android" android:host="OAuthActivity" />
         </intent-filter>
    </activity>
```

(6) 客户端根据临时令牌和用户授权码从服务提供方那里获取访问令牌 (Access Token)。

```
Uri uri=this.getIntent().getData();
RequestToken requestToken= OAuthConstant.getInstance().getRequestToken();
AccessToken
accessToken=requestToken.getAccessToken(uri.getQueryParameter("oauth ver
ifier"));
```

(7) 获得访问令牌后便可使用 API 接口获得和操作用户数据。

```
Weibo weibo=OAuthConstant.getInstance().getWeibo();
weibo.setToken(OAuthConstant.getInstance().getToken(),
OAuthConstant.getInstance().getTokenSecret());
String[] args = new String[2];
args[0]=OAuthConstant.getInstance().getToken();
args[1]=OAuthConstant.getInstance().getTokenSecret();
try {
    GetFollowers.main(args);//返回用户关注对象列表，并返回最新微博文章
} catch (Exception e) {
    e.printStackTrace();
}
```

在上述步骤中，weibo4android 是 XML 文件中定义的索引名，在步骤(5)的 XML 代码中，<data android:scheme="weibo4android" android:host="OAuthActivity" />部分的索引名是自定义的，只要与 Java 代码中的 URL 匹配即可。

在下面的内容中，将不再剖析 Android 版新浪微博的实现源码，而是以此为基础，讲解二次扩展开发的基本知识。

12.5.1　新浪微博图片缩放的开发实例

在 Android 开发过程中，有时会用到图片缩放效果，即点击图片时显示缩放按钮，过一会消失。下面将根据新浪微博的图片缩放原理编写演示代码以供参考。

(1) UI 布局文件的演示代码如下：

```
   <?xml version="1.0" encoding="utf-8"?>
   <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android: orientation="vertical"
          android:layout_width="fill_parent"
```

```
    android:layout height="fill parent"
    android:id="@+id/layout1"
       >

  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout width="fill parent"
        android:layout height="fill parent"
        android:id="@+id/rl"
          >

    <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout width="fill parent"
        android:layout height="fill parent"
        android:layout weight="19"
        android:scrollbars="none"
        android:fadingEdge="vertical"
        android:layout gravity="center"
        android:gravity="center"
        >

    <HorizontalScrollView
       android:layout height="fill parent"
       android:layout width="fill parent"
       android:scrollbars="none"
       android:layout gravity="center"
       android:gravity="center"
       android:id="@+id/hs"

          >
          <LinearLayout
                android: orientation="horizontal"
                android:layout width="fill parent"
                android:layout height="fill parent"
                android:id="@+id/layoutImage"
                android:layout gravity="center"
                android:gravity="center"
                >
                <ImageView
                   android:layout gravity="center"
                   android:gravity="center"
                      android:id="@+id/myImageView"
                      android:layout width="fill parent"
                      android:layout height="fill parent"
                      android:layout weight="19"
                      android:paddingTop="5dip"
                      android:paddingBottom="5dip"

                      />
          </LinearLayout>
    </HorizontalScrollView >
    </ScrollView>
```

```
            <ZoomControls android:id="@+id/zoomcontrol"
      android:layout width="wrap content" android:layout height=
      "wrap content" android:layout centerHorizontal="true"
      android:layout_alignParentBottom="true"
      >
    </ZoomControls>
       </RelativeLayout>

 </FrameLayout>
```

(2) 用 Java 编写主程序代码，代码如下：

```
package com.Johnson.image.zoom;
import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnKeyListener;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.os.Bundle;
import android.os.Handler;
import android.util.DisplayMetrics;
import android.util.Log;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.ZoomControls;
public class MainActivity extends Activity {
       /** Called when the activity is first created. */
  private final int LOADING IMAGE = 1;
  public static String KEY IMAGEURI = "ImageUri";
  private ZoomControls zoom;
  private ImageView mImageView;
  private LinearLayout layoutImage;
  private int displayWidth;
  private int displayHeight;
  /**图片资源*/
  private Bitmap bmp;
  /**宽的缩放比例*/
  private float scaleWidth = 1;
  /**高的缩放比例*/
  private float scaleHeight = 1;
  /**用来计数放大+1；缩小-1*/
  private int   zoomNumber=0;
```

```
/**点击屏幕显示缩放按钮，3 秒消失*/
private int showTime=3000;
RelativeLayout rl;
Handler mHandler = new Handler();
private Runnable task = new Runnable() {
  public void run() {
    zoom.setVisibility(View.INVISIBLE);
  }
};
      @Override
      public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.main);
  //showDialog(LOADING IMAGE);
  //图片是从网络上获取的话，需要加入滚动条
      bmp=BitmapFactory.decodeResource(getResources(), R.drawable.image);
  //removeDialog(LOADING IMAGE);
       initZoom();
}
  @Override
  protected Dialog onCreateDialog(int id) {
    switch (id) {
    case LOADING IMAGE: {
      final ProgressDialog dialog = new ProgressDialog(this);
      dialog.setOnKeyListener(new OnKeyListener() {
        @Override
        public boolean onKey(DialogInterface dialog, int keyCode,
            KeyEvent event) {
          if (keyCode == KeyEvent.KEYCODE BACK) {
            finish();
          }
          return false;
        }
      });
      dialog.setMessage("正在加载图片请稍后...");
      dialog.setIndeterminate(true);
      dialog.setCancelable(true);
      return dialog;
    }
    }
    return null;
  }
  public void initZoom() {

    /* 取得屏幕分辨率大小 */
    DisplayMetrics dm = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(dm);
    displayWidth = dm.widthPixels;
    displayHeight = dm.heightPixels;
    mImageView = (ImageView) findViewById(R.id.myImageView);
    mImageView.setImageBitmap(bmp);
```

```
    layoutImage = (LinearLayout) findViewById(R.id.layoutImage);
    mImageView.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        // TODO Auto-generated method stub
                              /**
                               * 在图片上和整个 View 上同时添加点击监听和捕捉屏幕
                               * 点击事件，来显示放大或缩小按钮
                               * */
        zoom.setVisibility(View.VISIBLE);
        mHandler.removeCallbacks(task);
        mHandler.postDelayed(task, showTime);
      }
    });
    layoutImage.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        // TODO Auto-generated method stub

        zoom.setVisibility(View.VISIBLE);
        mHandler.removeCallbacks(task);
        mHandler.postDelayed(task, showTime);
      }
    });

    zoom = (ZoomControls) findViewById(R.id.zoomcontrol);
    zoom.setIsZoomInEnabled(true);
    zoom.setIsZoomOutEnabled(true);
    // 图片放大
    zoom.setOnZoomInClickListener(new OnClickListener() {
      public void onClick(View v) {
        big();
      }
    });
    // 图片缩小
    zoom.setOnZoomOutClickListener(new OnClickListener() {

      public void onClick(View v) {
        small();
      }

    });
    zoom.setVisibility(View.VISIBLE);
    mHandler.postDelayed(task, showTime);

  }

  @Override
  public boolean onTouchEvent(MotionEvent event) {
```




```
  // TODO Auto-generated method stub
      /**
             * 在图片上和整个 View 上同时添加点击监听和捕捉屏幕
             * 点击事件，来显示放大或缩小按钮
             * */
  zoom.setVisibility(View.VISIBLE);
  mHandler.removeCallbacks(task);
  mHandler.postDelayed(task, showTime);
  return false;
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  // TODO Auto-generated method stub
  super.onKeyDown(keyCode, event);

  return true;
}

/* 图片缩小的 method */
private void small() {
  --zoomNumber;
  int bmpWidth = bmp.getWidth();
  int bmpHeight = bmp.getHeight();

  Log.i("","bmpWidth = " + bmpWidth + ", bmpHeight = " + bmpHeight);

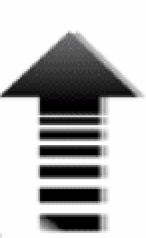
  /* 设置图片缩小的比例 */
  double scale = 0.8;
  /* 计算出这次要缩小的比例 */
  scaleWidth = (float) (scaleWidth * scale);
  scaleHeight = (float) (scaleHeight * scale);
  /* 产生 reSize 后的 Bitmap 对象 */
  Matrix matrix = new Matrix();
  matrix.postScale(scaleWidth, scaleHeight);
  Bitmap resizeBmp = Bitmap.createBitmap(bmp, 0, 0, bmpWidth, bmpHeight,
     matrix, true);
  mImageView.setImageBitmap(resizeBmp);

  /* 限制缩小尺寸 */
  if ((scaleWidth * scale * bmpWidth < bmpWidth / 4
     || scaleHeight * scale * bmpHeight > bmpWidth /4
     || scaleWidth * scale * bmpWidth > displayWidth / 5
     || scaleHeight * scale * bmpHeight > displayHeight /
        5)&&(zoomNumber==-1) ){

  zoom.setIsZoomOutEnabled(false);

  } else {

  zoom.setIsZoomOutEnabled(true);
```



```
    }

    zoom.setIsZoomInEnabled(true);
    System.gc();
  }

  /* 图片放大的method */
  private void big() {
    ++zoomNumber;
    int bmpWidth = bmp.getWidth();
    int bmpHeight = bmp.getHeight();

    /* 设置图片放大的比例 */
    double scale = 1.25;
    /* 计算这次要放大的比例 */
    scaleWidth = (float) (scaleWidth * scale);
    scaleHeight = (float) (scaleHeight * scale);
    /* 产生reSize后的Bitmap对象 */
    Matrix matrix = new Matrix();
    matrix.postScale(scaleWidth, scaleHeight);
    Bitmap resizeBmp = Bitmap.createBitmap(bmp, 0, 0, bmpWidth, bmpHeight,
        matrix, true);
    mImageView.setImageBitmap(resizeBmp);
    /* 限制放大尺寸 */
    if (scaleWidth * scale * bmpWidth > bmpWidth * 4
        || scaleHeight * scale * bmpHeight > bmpWidth * 4
        || scaleWidth * scale * bmpWidth > displayWidth * 5
        || scaleHeight * scale * bmpHeight > displayHeight * 5) {

      zoom.setIsZoomInEnabled(false);

    } else {

      zoom.setIsZoomInEnabled(true);

    }

    zoom.setIsZoomOutEnabled(true);

  System.gc();
  }

}
```

12.5.2　添加分享到新浪微博

现在很多平台都开放了，并且提供了相应的接口。在过去你浏览论坛或者博客的时候，论坛或博客都需要自己的账号，但是现在你会发现都有一个“用新浪微博登录”、

“用 QQ 账号登录”等的字样。这样经过授权以后你就可以用新浪或者腾讯的账号登录到论坛或者博客了，这确实是挺方便的事情，可以直接为你的社区带来用户流量。

最近开发的应用都涉及分享的功能，Android 系统有内置的分享功能，但是内置的分享只有在你安装该应用的时候才会被显示在列表中，下面是 Android 系统内置的分享，如图 12-4 所示。



图 12-4　Android系统内置的分享

选择图 12-4 中的“分享”选项后即可看到“新浪微博”，这个是笔者自己添加的。意思就是说：如果安装了“新浪微博”移动端，就用系统自己的分享。如果没有安装该应用，则需自行添加分享到“新浪微博”的功能。下面我们看看这个列表是怎么加载的：

```
Intent intent = new Intent(Intent.ACTION SEND);
intent.setType("text/plain");
ShareAdapter mAdapter = new ShareAdapter(mContext, intent);
//对话框的适配器
public class ShareAdapter extends BaseAdapter {
    private final static String PACKAGENAME = "com.sina.weibo";
    private Context mContext;
    private PackageManager mPackageManager;
    private Intent mIntent;
    private LayoutInflater mInflater;
    private List<ResolveInfo> mList;
    private List<DisplayResolveInfo> mDisplayResolveInfoList;
    public ShareAdapter(Context context, Intent intent) {
        mContext = context;
        mPackageManager = mContext.getPackageManager();
        mIntent = new Intent(intent);
        mInflater = (LayoutInflater)mContext.getSystemService
            (Context.LAYOUT INFLATER SERVICE);
        mList = mContext.getPackageManager().queryIntentActivities
                (intent, PackageManager.MATCH DEFAULT ONLY);
        // 排序
        ResolveInfo.DisplayNameComparator comparator = new
            ResolveInfo.DisplayNameComparator(mPackageManager);
        Collections.sort(mList, comparator);
```

```
        mDisplayResolveInfoList = new ArrayList<DisplayResolveInfo>();
        if (mList == null || mList.isEmpty()) {
            mList = new ArrayList<ResolveInfo>();
        }
        final int N = mList.size();
        for (int i = 0; i < N; i++) {
            ResolveInfo ri = mList.get(i);
            CharSequence label = ri.loadLabel(mPackageManager);
            DisplayResolveInfo d = new DisplayResolveInfo(ri, null, null,
                label, null);
            mDisplayResolveInfoList.add(d);
        }
        //考虑是否已安装新浪微博，如果没有，则自行添加
        if(!isInstallApplication(mContext, PACKAGENAME)){
            Intent i = new Intent(mContext, ShareActivity.class);
            Drawable d = mContext.getResources().getDrawable(R.drawable.sina);
            CharSequence label = mContext.getString(R.string.about sina weibo);
            DisplayResolveInfo dr = new DisplayResolveInfo(null, i, null,
               label, d);
            mDisplayResolveInfoList.add(0, dr);
        }
    }
    @Override
    public int getCount() {
        return mDisplayResolveInfoList.size();
    }

    @Override
    public Object getItem(int position) {
        return mDisplayResolveInfoList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }
    @Override
   public View getView(int position, View convertView, ViewGroup parent)
{
        View item;
        if(convertView == null) {
            item = mInflater.inflate(R.layout.share item, null);
        } else {
            item = convertView;
        }
        DisplayResolveInfo info = mDisplayResolveInfoList.get(position);

        ImageView i = (ImageView) item.findViewById(R.id.share item icon);
        if(info.mDrawable == null){
            i.setImageDrawable(info.mResoleInfo.loadIcon(mPackageManager));
        }else{
```

```
            i.setImageDrawable(info.mDrawable);
        }

        TextView t = (TextView) item.findViewById(R.id.share item text);
        t.setText(info.mLabel);
        return item;
    }

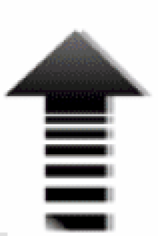
    public ResolveInfo getResolveInfo(int index){
        if(mDisplayResolveInfoList == null){
            return null;
        }
        DisplayResolveInfo d = mDisplayResolveInfoList.get(index);
        if(d.mResoleInfo == null){
            return null;
        }
        return d.mResoleInfo;
    }

    //返回跳转 intent
    public Intent getIntentForPosition(int index) {
        if(mDisplayResolveInfoList == null){
            return null;
        }
        DisplayResolveInfo d = mDisplayResolveInfoList.get(index);
        Intent i = new Intent(d.mIntent == null ? mIntent : d.mIntent);
        i.addFlags(Intent.FLAG ACTIVITY FORWARD RESULT |
           Intent.FLAG ACTIVITY PREVIOUS IS TOP);
        if(d.mResoleInfo != null){
            ActivityInfo a = d.mResoleInfo.activityInfo;
            i.setComponent(new ComponentName
                (a.applicationInfo.packageName, a.name));
        }
        return i;
    }

    //检查是否安装该 APP
    boolean isInstallApplication(Context context, String packageName){
        try {
            mPackageManager
                .getApplicationInfo(packageName,
                PackageManager.GET UNINSTALLED PACKAGES);
            return true;
        } catch (NameNotFoundException e) {
            return false;
        }

    }

    /**
     * 打包数据 vo
```



```
     * @author Administrator
     */
    class DisplayResolveInfo {
        private Intent mIntent;
        private ResolveInfo mResoleInfo;
        private CharSequence mLabel;
        private Drawable mDrawable;

        DisplayResolveInfo(ResolveInfo resolveInfo, Intent intent,
                CharSequence info, CharSequence label, Drawable d) {
            this.mIntent = intent;
            this.mResoleInfo = resolveInfo;
            this.mLabel = label;
            this.mDrawable = d;
        }
    }
}
```

以上是加载弹出对话框的数据适配器，如果系统已经安装了，则直接读取系统的分享，没有安装，则添加。当你点击分享微博的时候，就需要一系列的验证和授权了，这边采用的机制是先获取 requestToken，再通过 requestToken 获取 AccessToken，然后才可以分享微博。开始的时候笔者也是从新浪官方的现在的 SDK 不知道是 1.0 还是 2.0，但是始终都不能发送微博，诡异的是用官方的 SDK 可以认证完成，并能够获取微博内容，但是死活发不了微博，郁闷好几天。但是看到它的官方论坛里面有那么多的受害者，我表示沉默，一个偌大的公司提供一个接口居然成这样。现在我用的这个 SDK 版本里面的 Weibo.java 有很多其他的方法，如获取用户信息、收藏微博等，大家可以自己看看。

接下来是在你点击“分享到微博”的时候进行认证用户。首先说明一下这里的认证是读取新浪提供的页面，显示的界面如图 12-4 右图，下面是部分代码：

```
Weibo weibo = new Weibo();
RequestToken requestToken =
weibo.getOAuthRequestToken("yunmai://ShareActivity");
<span style="color:#e53333;">//与配置中对应</span> Log.i(TAG, "token:" +
requestToken.getToken() + ",tokenSecret:" +
requestToken.getTokenSecret());
OAuthConstant.getInstance().setRequestToken(requestToken);
Uri uri = Uri.parse(requestToken.getAuthenticationURL() +
"&display=mobile");
url = uri.toString();
```

上面的地址 URL 就是你请求新浪提供的登录界面的地址，这里会涉及 Webview 的使用。在 Android 中 Webview 其实就是一个小型浏览器，功能很强大，强大到可以执行脚本。有了地址可以通过 webview.loadurl(URL)请求登录界面。也有很多网友可能想自己设计一个登录界面，但是新浪官方有说明，通过 getXauthAccessToken 方式认证是可以自行设计登录界面的，其他认证方式是不能够自行设计的。在你单击“授权”按钮时需要跳转到我们自己的 Activity 这里的配置是需要在 androidmanifest.xml 中进行配置，比如我这边跳转的是 shareactivity.java。

```
<activity
          android:name="cn.yunmai.cclauncher.ShareActivity"
          android:screenOrientation="portrait" >
          <intent-filter>
             <action android:name="android.intent.action.VIEW" />
             <category android:name="android.intent.category.DEFAULT" />
             <category android:name="android.intent.category.BROWSABLE" />
             <data android:host="ShareActivity" android:scheme="yunmai" />
          </intent-filter>
      </activity>
```

需要注意的是，在<data>标签中的内容需要和显示授权窗口中的weibo.getOAuthRequestToken("yunmai://ShareActivity")相对应。在授权完成之后就会跳转到自己定义的 Activity，授权完成之后就可发微博了，跳转之后的 Activity 现在就比较清楚明白了，简单点就可以放一个“发送”按钮，一个 editText 就可以了。在你单击“发送”按钮的时候，你要获取刚才授权成功的 RequestToken，然后再获取 accessToken，最后发送微博。需要注意的是，RequestToken 只需获取一次，然后保存你的 accessToekn，这个是你每次都需要使用的口令，这里就可以优化一下体验了。单击发送按钮的操作是：

```
Uri uri = this.getIntent().getData();
RequestToken requestToken =
OAuthConstant.getInstance().getRequestToken();
AccessToken accessToken =
requestToken.getAccessToken(uri.getQueryParameter("oauth verifier"));
saveAccessToken(accessToken);//保存 accessToken
Log.i(TAG, "oauth verifier:" + uri.getQueryParameter("oauth verifier") +
    ",Token" + accessToken.getToken() + ",TokenSecret:" +
     accessToken.getTokenSecret());
OAuthConstant.getInstance().setAccessToken(accessToken);
```

这里所执行的操作是获取授权之后的 accessToken，然后发送微博：

```
Weibo weibo = OAuthConstant.getInstance().getWeibo();
weibo.setToken(OAuthConstant.getInstance().getToken(),
OAuthConstant.getInstance().getTokenSecret());
Status s = weibo.updateStatus(mEdit.getText().toString());
```

status 返回了一些详细的信息，有发送时间和用户 ID 等，这就样就完成了分享功能。

12.5.3 通过Json对象登录新浪微博

我们可以引用新浪开发包中的各种类，在 Android 中通过 JSON 对象的方式登录新浪微博。在下面的代码中，1 代表登录成功，0 代表登录失败，并通过方法 verifyCredentials()请求新浪微博服务器返回 Json 对象。

```
package com.sfc.ui;

import java.util.ArrayList;
import java.util.List;
```

```
import com.sfc.ui.adapter.LoginListAdapter;

import weibo4j.User;                              //这是新浪开发包中的实体类
import weibo4j.Weibo;                             //这是新浪开发包中的类
import weibo4j.WeiboException;                    //这是新浪开发包中的类

import android.app.Activity;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;

public class LoginActivity extends Activity implements Runnable {
 private Button loginButton;
 private ListView listView;
 private ProgressDialog loginDialog;
 private Thread loginThread;
 private Handler handler;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.login);
  loginButton = (Button)findViewById(R.id.loginButton);
  List<String> list = new ArrayList<String>();
  list.add("随便看看");
  list.add("推荐用户");
  list.add("热门转发");
  listView = (ListView)findViewById(R.id.listView);
  loginThread = new Thread(this);

  handler = new Handler(){
          //1 代表登录成功，0 代表登录失败
        public void handleMessage(Message msg) {
         loginDialog.cancel();
            switch (msg.what) {
          case 1:
           Toast.makeText(LoginActivity.this, "登录成功 ", 3000).show();
             break;
          case 0:
           Toast.makeText(LoginActivity.this, "登录失败", 3000).show();
           break;
          }
        };
    };
```

```
    listView.setAdapter(new LoginListAdapter(this,list));
    loginButton.setOnClickListener(new OnClickListener(){
     public void onClick(View v) {
      loginDialog = new ProgressDialog(LoginActivity.this);
      loginDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
      loginDialog.setMessage("登录服务器");
      loginDialog.show();
      loginThread.start();
     }
    });
   }
  public void run() {
    Log.e("loginThread","start");
    Weibo weibo = new Weibo("XXX@sina.com","XXX");   //新浪微博用户名和密码
    weibo.setHttpConnectionTimeout(5000);
    Message  msa = new Message();
    try {
     User user = weibo.verifyCredentials();
           //该方法会请求新浪微博服务器返回 Json 对象
     msa.what=1;
    } catch (WeiboException e) {
     msa.what=0;
    }
   }
}
```

12.5.4 实现OAuth认证

OAuth 协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同之处是 OAuth 的授权不会使第三方触及到用户的账号信息(如用户名与密码)，即第三方无须使用用户的用户名与密码就可以申请获得该用户资源的授权，因此 OAuth 是安全的。

新浪微博为了实现自身的安全性，采用了 OAuth 协议认证方式。虽然下面的一段代码比较简单，但是实现了新浪微博的 OAuth 认证。

```
System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
System.setProperty("weibo4j.oauth.consumerSecret",
Weibo.CONSUMER_SECRET);
    Weibo weibo = new Weibo();
    // set callback url, desktop app please set to null
    // http://callback_url?oauth_token=xxx&oauth_verifier=xxx
    //1.根据 app key 第三方应用向新浪获取 requestToken
    RequestToken requestToken = weibo.getOAuthRequestToken();
    System.out.println("1.......Got request token 成功");
    System.out.println("Request token: "+ requestToken.getToken());
    System.out.println("Request token secret: "+ requestToken.getTokenSecret());
    AccessToken accessToken = null;
   //2.用户从新浪获取 verifier_code，如果是 Android 或 Iphone 应用，可以 callback
```

```
     =json&userId=xxs&password=XXX
    System.out.println("Open the following URL and grant access to your account:");
    System.out.println(requestToken.getAuthorizationURL());
    BareBonesBrowserLaunch.openURL(requestToken.getAuthorizationURL());
   //3.用户输入验证码授权信任第三方应用
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    while (null == accessToken) {
     System.out.print("Hit enter when it's done.[Enter]:");
     String pin = br.readLine();
     System.out.println("pin: " + br.toString());
     try{
      //4.通过传递 requestToken 和用户验证码获取 AccessToken
         accessToken = requestToken.getAccessToken(pin);
     } catch (WeiboException te) {
         if(401 == te.getStatusCode()){
          System.out.println("Unable to get the access token.");
         }else{
          te.printStackTrace();
         }
     }
    }
    System.out.println("Got access token.");
    System.out.println("Access token: "+ accessToken.getToken());
    System.out.println("Access token secret: "+ accessToken.getTokenSecret());
   //使用 AccessToken 来操作用户的所有接口
   /* Weibo weibo=new Weibo();
    以后就可以用下面 accessToken 访问用户的资料了
    * weibo.setToken(accessToken.getToken(), accessToken.getTokenSecret());
    //发布微博
Status status = weibo.updateStatus("test message6 ");
System.out.println("Successfully updated the status to ["
    + status.getText() + "].");
    try {
Thread.sleep(3000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}*/
    System.exit(0);
} catch (WeiboException te) {
    System.out.println("Failed to get timeline: " + te.getMessage());
    System.exit( -1);
} catch (IOException ioe) {
    System.out.println("Failed to read the system input.");
    System.exit( -1);
}
```

Android

第 13 章 流量统计系统

在 Android 网络项目编程应用中，流量统计是最常见的一种应用。通过流量统计功能，可以及时了解手机使用网络流量的状况。在本章的内容中，将详细讲解在 Android 系统中实现流量统计功能的基本知识，介绍了实现思路，为步入本书后面知识的学习打下基础。

13.1　流量统计基础

流量统计功能十分重要。通过流量统计可以及时获取我们使用过的网络流量，避免超出各种包月流量的限制。本节将简要讲解在 Android 中实现流量统计的基本知识。

13.1.1　TrafficStats类

对于 Android 流量统计功能来说，从 2.2 版中开始加入了 TrafficStats 类，通过此类可以轻松获取 Android 手机的流量。在具体实现时，TrafficStats 类是通过读取 Linux 提供的文件对象系统类型的文本进行解析的。在 android.net.TrafficStats 类中提供了多种静态方法，通过这些方法可以直接调用获取流量信息，返回类型均为 long 型，如果返回“-1”，则代表 UNSUPPORTED，即当前设备不支持统计。

TrafficStats 类中的统计包括所有网络接口、Mobile 接口和 UID 网络接口的字节发送和接收，以及网络数据包的发送和接收等。其继承关系如下：

```
public class TrafficStatsextends Object
java.lang.Object
android.net.TrafficStats
```

功能：获取通过 Mobile 接口发送的数据包总数。

返回值：数据包总数。如果本设备不支持统计，将返回 UNSUPPORTED。

1. TrafficStats类的常量

public static final int UNSUPPORTED：返回值表示该设备不支持统计。常量值是-1(0xffffffff)。

2. TrafficStats类的公共方法

TrafficStats 类的公共方法如下。

- public static long getMobileRxBytes()：获取通过 Mobile 接口接收到的字节总数，不包含 Wi-Fi。返回值是字节总数。如果本设备不支持统计，将返回 UNSUPPORTED。
- public static long getMobileRxPackets()：获取通过 Mobile 接口接收到的数据包总数。返回值是数据包总数。如果本设备不支持统计，将返回 UNSUPPORTED。
- public static long getMobileTxBytes()：获取通过 Mobile 接口发送的字节总数，返回值是字节总数。如果本设备不支持统计，将返回 UNSUPPORTED。
- public static long getMobileTxPackets()：获取通过 Mobile 接口发送的数据包总数，返回值是数据包总数。如果本设备不支持统计，将返回 UNSUPPORTED。
- public static long getTotalRxBytes()：获取通过所有网络接口接收到的字节总数，包含 Mobile 和 Wi-Fi 等。返回值是字节总数。如果本设备不支持统计，将返回 UNSUPPORTED。



- public static long getTotalRxPackets()：获取通过所有网络接口接收到的数据包总数，包含 Mobile 和 Wi-Fi 等。返回值是数据包总数。如果本设备不支持统计，将返回 UNSUPPORTED。
- public static long getTotalTxBytes()：获取通过所有网络接口发送的字节总数，包含 Mobile 和 Wi-Fi 等。返回值是字节总数。如果本设备不支持统计，将返回 UNSUPPORTED。
- public static long getTotalTxPackets()：获取通过所有网络接口发送的数据包总数，包含 Mobile 和 Wi-Fi 等。返回值是数据包总数。如果本设备不支持统计，将返回 UNSUPPORTED。
- public static long getUidRxBytes(int uid)：获取通过 UID 网络接口接收到的字节数，统计包含所有网络接口。参数 uid 表示待检查的进程的 UID。返回值是字节数。
- public static long getUidTxBytes (int uid)：获取通过 UID 网络接口发送的字节数，统计包含所有网络接口。参数 uid 表示待检查的进程的 UID。返回值是字节总数，如果本设备不支持统计，将返回 UNSUPPORTED。

注意： 类 TrafficStats 通常通过以下三个方法查看具体流量。

- getMobileRxBytes()
- getTotalRxBytes()
- getUidRxBytes()或 getUidTxBytes()

Android 的流量统计功能是通过 ndk 调用以下文件实现的。

- 发送包：/sys/class/net/rmnet0/statistics/tx_packets。
- 接收包：/sys/class/net/rmnet0/statistics/rx_packets。
- 发送字节：/sys/class/net/rmnet0/statistics/tx_bytes。
- 接收字节：/sys/class/net/rmnet0/statistics/rx_bytes 或/proc/self/net/dev。

在具体测试时会发现，各进程 getUidRxBytes 的值总与 MobileRxBytes 不一致。经过查看 getUidRxBytes()和 getUidTxBytes()的 native(本地)代码后，会发现此方法通过读取/proc/uid_stat/%d/tcp_rcv 和/proc/uid_stat/%d/tcp_snd 文件来获取流量，其中%d 为进程 UID。这两个文件为非标准 Linux 内核文件，由 Android 内核层/kernel/net/Socket.c 的函数__sock_sendmsg()负责写入。用户层套接字通信在内核层最终会调用此函数，包括本地套接字和网络套接字。所以根据 TrafficStats.getUidRxBytes() 或 getUidTxBytes()获取的流量，不但包括了网络流量，而且也包括了本地流量。

而 MobileRxBytes()读取的是如下数据。

- sys/class/net/rmnet0/statistics/rx_bytes
- sys/class/net/ppp0/statistics/rx_bytes

13.1.2　Android流量统计的基本思路

在 Android 2.2 版本以后，主要使用类 TrafficStats 中的方法来实现流量统计。由于在应用项目中需要用到监控各个应用的流量，所以基本思路是：先获取手机中所有具有联网

权限的应用，并且以列表形式显示出来，然后选择相应的应用，即可对其进行流量监控。这里用到的便是方法 getUidRxBytes(int uid)和方法 getUidTxBytes(int uid)。在获取手机中所有具有联网权限的应用之前，把手机中所有应用的流量使用情况都显示了出来。事实证明，连拨号器操作都会耗费流量。由此可见，用这两个类方法获取的流量包括本地流量，所以就会出现连没有联网权限的应用都发送和接收字节数这种情况。

下面是一段根据上述思路实现流量监控的代码。

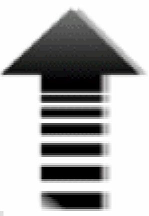
```
public void dingshi(){

                mdb.openDB();//打开数据库

                runnableapp=new Runnable(){ //线程对象
                @Override
                public void run() {
                    while(flag==0){//用于停止线程时的判断
                        try {
                            Thread.sleep(1000);//每 1000ms 进行流量监控
                        } catch (InterruptedException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                    int i=0;
                    Log.v("app", "id---->"+Thread.currentThread().getId());
                    try{
                    for (i=0;i<listuser.size();i++) {//listuser 想要监控的
                        应用，只要列表不再重新选择，则获取应用的顺序一定
                        Log.v("app", "开始"+Thread.currentThread().getId());
                        uid=Integer.parseInt(listuser.get(i));
                            //列表中第 I 个选择的应用的 uid
                        appinfo=listappinfo.get(i);

                         Calendar calendar = Calendar.getInstance();
                           int month=calendar.get(Calendar.MONTH)+1;
                           int day=calendar.get(Calendar.DAY OF MONTH);
                           int hour=calendar.get(Calendar.HOUR OF DAY);
                           int minute=calendar.get(Calendar.MINUTE);
                           int second=calendar.get(Calendar.SECOND);
                           float mill=calendar.get(Calendar.MILLISECOND);
                           String time= calendar.get(Calendar.YEAR) + "年"
                                + month + "月"
                                + day + "日"
                                + hour + "时"
                                + minute + "分"
                                +second+"秒"
                                +mill+"毫秒";
time02[i]=day*24*60*60+hour*60*60+minute*60+second+mill/1000;
    //单位是秒，新获取的时间，肯定大于前面获取的时间

                      System.out.println("现在的时间"+i+"--->"+time02[i]);
```



```
                              System.out.println("之前的时间"+i+"--->"+time01[i]);

                               recv[i]=TrafficStats.getUidRxBytes(uid);
                               if(recv[i]>0) {
                               recv[i]=recv[i]/(1024*1024);//单位为 MB
                               }else{
                                recv[i]=0;
                               }
                               tran[i]=TrafficStats.getUidTxBytes(uid);
                               if(tran[i]>0){
                               tran[i]=tran[i]/(1024*1024);
                               }
                               else{
                                tran[i]=0;
                               }
                               totalapp[i]=recv[i]+tran[i];
recvrate[i]=((recv[i]-recv01[i])*1024*1024)/(time02[i]-time01[i]);
    //单位为 B/s，均为瞬时速率
tranrate[i]=((tran[i]-tran01[i])*1024*1024)/(time02[i]-time01[i]);
                               totalapprate[i]=((totalapp[i]-
totalapp01[i])*1024*1024)/(time02[i]-time01[i]);

                               System.out.println(appinfo+i+"---->" +"接收速率---->"
                               +recvrate[i]+ "发送速率---->" + tranrate[i]+"total
                               速率---->"+totalapprate[i]);
                               System.out.println("时间间隔"+i+"---->"+(time02[i]-
                                    time01[i]));

                               recv01[i]=recv[i];//给全局变量赋新值
                                tran01[i]=tran[i];
                                totalapp01[i]=totalapp[i];
                              time01[i]=time02[i];


                               System.out.println("uid=" + uid +"---->"+ "recv="
                                   + recv[i]);
                               System.out.println("uid=" + uid +"---->"+ "tran="
                                   + tran[i]);
                               System.out.println("uid=" + uid +"---->"+ totalapp[i]);
                               System.out.println("appinfo=" + appinfo );
                               try{
                               mdb.addTrafficData(time, appinfo, String.valueOf(uid),
                                 String.valueOf(tran[i]),String.valueOf(recv[i]),String.
                                 valueOf(totalapp[i]),String.valueOf(tranrate[i]),
                                 String.valueOf(recvrate[i]),String.valueOf
                                 (totalapprate[i]));
                              }catch(Exception e){
                                  System.out.println("----->app 出现异常");
                              }
                               Log.v("app", "结束"+Thread.currentThread().getId());
                          }
```

```
                    }catch(Exception e){
                        System.out.println("异常");
                    }


                            }
                }
                            };
                            Thread AppThread = new Thread(runnableapp);
                            AppThread.start();


                    }
```

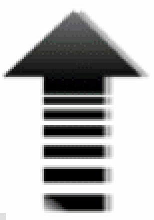
13.1.3 读取Linux内核获取流量信息

Android 手机流量信息系统是基于 Linux 内核的，记录在/proc/self/net/dev 文件中。dev 文件的格式如下：

```
Microsoft Windows XP [版本 5.1.2600]
 (C) 版权所有 1985-2001 Microsoft Corp.
D:/Program Files/Java/sdk/android-sdk-windows/tools>adb shell
# cd proc
cd proc
# cd net
cd net
# cat dev
cat dev
Inter-| Receive | Transmit face |bytes packets errs drop fifo frame
compressed multicast|bytes packe ts errs drop fifo colls carrier
compressed
lo: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
eth0: 7069733 86239 0 0 0 0 0 0 12512463 741 79 0 0 0 0 0 0
tunl0: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
gre0: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 #
```

可以通过读取 dev 文件来获取流量信息，例如下面的代码：

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.util.Calendar;
import org.apache.http.util.EncodingUtils;
import android.app.Service;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
```



```
import android.widget.Toast;

public class mService1 extends Service
{
    private Handler objHandler = new Handler();
    private int intCounter = 0;
    private int mHour;
    private int mMinute;
    private int mYear;
    private int mMonth;
    private int mDay;
    private String mdate;

    final public String DEV_FILE = "/proc/self/net/dev";// 系统流量文件
    String[] ethdata = { "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
            "0", "0", "0", "0", "0" };
    String[] gprsdata = { "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
            "0", "0", "0", "0", "0", "0" };
    String[] wifidata = { "0", "0", "0", "0", "0", "0", "0", "0", "0", "0",
            "0", "0", "0", "0", "0", "0" };
    String data = "0,0,0,0,0,0,0,0,0,0,0,0";// 对应 on.txt 里面的格式

    final String ETHLINE = "  eth0";
        // eth 是以太网信息，tiwlan0 是 Wi-Fi，rmnet0 是 GPRS
    final String GPRSLINE = "rmnet0";
    //转载时此处为 tiwlan0， 但发现在我的 2.1 的手机上为 wlan0
    final String WIFILINE = " wlan0";
    final String TEXT_ENCODING = "UTF-8";
    final public String ONPATH = "/data/data/zy.dnh/on.txt";
    final public String LOGPATH = "/data/data/zy.dnh/log.txt";

    private Runnable mTasks = new Runnable()
    {
        public void run()// 运行该服务执行此函数
        {
            refresh();
            intCounter++;
            // DisplayToast("Counter:"+Integer.toString(intCounter));
            objHandler.postDelayed(mTasks, 3000);// 每 3000ms 执行一次
        }
    };

    @Override
    public void onStart(Intent intent, int startId)
    {
        // TODO Auto-generated method stub
        objHandler.postDelayed(mTasks, 0);
        super.onStart(intent, startId);
    }

    @Override
```

```
public void onCreate()
{
    // TODO Auto-generated method stub
    super.onCreate();
}

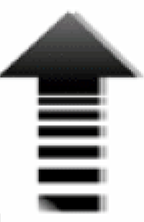
@Override
public IBinder onBind(Intent intent)
{
    // TODO Auto-generated method stub
    return null;
}

@Override
public void onDestroy()
{
    // TODO Auto-generated method stub
    /*  */
    objHandler.removeCallbacks(mTasks);
    super.onDestroy();
}

public void DisplayToast(String str)
{
    Toast.makeText(this, str, Toast.LENGTH SHORT).show();
}

public void readdev()
{
    FileReader fstream = null;
    try {
        fstream = new FileReader(DEV FILE);
    }
    catch (FileNotFoundException e) {
        DisplayToast("Could not read " + DEV FILE);
    }

    BufferedReader in = new BufferedReader(fstream, 500);
    String line;
    String[] segs;
    String[] netdata;
    int count = 0;
    int k;
    int j;
    try {
        while ((line = in.readLine()) != null) {
            segs = line.trim().split(":");
            if (line.startsWith(ETHLINE))
            {
                netdata = segs[1].trim().split(" ");
                for (k = 0, j = 0; k < netdata.length; k++)
```



```
                {
                    if (netdata[k].length() > 0)
                    {
                        ethdata[j] = netdata[k];
                        j++;
                    }
                }
            }
            else if (line.startsWith(GPRSLINE))
            {
                netdata = segs[1].trim().split(" ");
                for (k = 0, j = 0; k < netdata.length; k++)
                {
                    if (netdata[k].length() > 0)
                    {
                        gprsdata[j] = netdata[k];
                        j++;
                    }
                }
            }
            else if (line.startsWith(WIFILINE))
            {
                netdata = segs[1].trim().split(" ");
                for (k = 0, j = 0; k < netdata.length; k++)
                {
                    if (netdata[k].length() > 0)
                    {
                        wifidata[j] = netdata[k];
                        j++;
                    }
                }
            }
            count++;
        }

        fstream.close();
    }
    catch (IOException e) {
        DisplayToast(e.toString());
    }

}

public String getinfo(String path)
{
    File file;
    String str = "";
    FileInputStream in;
    try {
        // 打开文件 file 的 InputStream
        file = new File(path);
```

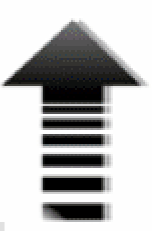



```
        in = new FileInputStream(file);
        // 将文件内容全部读入到byte数组
        int length = (int) file.length();
        byte[] temp = new byte[length];
        in.read(temp, 0, length);
        // 将byte数组用UTF-8编码并存入display字符串中
        str = EncodingUtils.getString(temp, TEXT ENCODING);
        // 关闭文件file的InputStream
        in.close();
    }
    catch (IOException e) {
        DisplayToast(e.toString());
    }
    return str;
}

public void writefile(String str, String path)
{
    File file;
    FileOutputStream out;
    try {
        // 创建文件
        file = new File(path);
        file.createNewFile();
        // 打开文件file的OutputStream
        out = new FileOutputStream(file);
        String infoToWrite = str;
        // 将字符串转换成byte数组写入文件
        out.write(infoToWrite.getBytes());
        // 关闭文件file的OutputStream
        out.close();
    } catch (IOException e) {
        // 将出错信息打印到Logcat
        DisplayToast(e.toString());
    }
}

public void refresh()
{
    readdev();// 读取本次开机之后直到当前系统的总流量
    data = ethdata[0] + "," + ethdata[1] + "," + ethdata[8] + ","
            + ethdata[9] + ","
            + gprsdata[0] + "," + gprsdata[1] + "," + gprsdata[8] + ","
            + gprsdata[9] + ","
            + wifidata[0] + "," + wifidata[1] + "," + wifidata[8] + ","
            + wifidata[9];

    String onstr = getinfo(ONPATH);     // 读取on.txt记录到onstr里
    String ondata[] = onstr.split(","); // 将onstr各项分离放到ondata里
    // 计算增量
```



```
int[] delta = new int[12];
delta[0] = Integer.parseInt(ethdata[0]) - Integer.parseInt(ondata[0]);
delta[1] = Integer.parseInt(ethdata[1]) - Integer.parseInt(ondata[1]);
delta[2] = Integer.parseInt(ethdata[8]) - Integer.parseInt(ondata[2]);
delta[3] = Integer.parseInt(ethdata[9]) - Integer.parseInt(ondata[3]);
delta[4] = Integer.parseInt(gprsdata[0]) - Integer.parseInt(ondata[4]);
delta[5] = Integer.parseInt(gprsdata[1]) - Integer.parseInt(ondata[5]);
delta[6] = Integer.parseInt(gprsdata[8]) - Integer.parseInt(ondata[6]);
delta[7] = Integer.parseInt(gprsdata[9]) - Integer.parseInt(ondata[7]);
delta[8] = Integer.parseInt(wifidata[0]) - Integer.parseInt(ondata[8]);
delta[9] = Integer.parseInt(wifidata[1]) - Integer.parseInt(ondata[9]);
delta[10] = Integer.parseInt(wifidata[8])- Integer.parseInt(ondata[10]);
delta[11] = Integer.parseInt(wifidata[9])- Integer.parseInt(ondata[11]);

// 读取 log.txt
// 获取当前时间
final Calendar c = Calendar.getInstance();
mYear = c.get(Calendar.YEAR);           // 获取当前年份
mMonth = c.get(Calendar.MONTH) + 1;     // 获取当前月份
mDay = c.get(Calendar.DAY_OF_MONTH);    // 获取当前月份的日期号码
mHour = c.get(Calendar.HOUR_OF_DAY);    // 获取当前的小时数
mMinute = c.get(Calendar.MINUTE);       // 获取当前的分钟数
mdate = mYear + "-" + mMonth + "-" + mDay;
String text = getinfo(LOGPATH);     // 将 log.txt 的内容读到 text 字符串中
String[] line = text.split("/n");
String today = line[line.length - 1];  // 获得今日已记录流量
String[] beToday = today.split(",");

// 检查文件最后一行是否为今天的流量记录信息
if (!beToday[0].equals(mdate))
    // 如果文件只有一行，表明目前日志为空，将当前日期加入
    // 判断今日流量是否已经记录，如果今日流量没有记录
{
    text = text + mdate + ",0,0,0,0,0,0,0,0,0,0,0,0/n";
    writefile(text, LOGPATH);
    line = text.split("/n");
    today = line[line.length - 1];// 获得今日已记录流量
    beToday = today.split(",");
}
int i;
// 处理今日流量
int[] newTodaydata = new int[12];// 表示今日流量
String newtoday = mdate;
for (i = 0; i <= 11; i++)
{
   newTodaydata[i] = Integer.parseInt(beToday[i + 1]) + delta[i];
    newtoday = newtoday + "," + newTodaydata[i];
}
newtoday = newtoday + "/n";
String[] beTotal = line[0].split(",");
int[] newTotaldata = new int[12];// 表示总流量数值
```

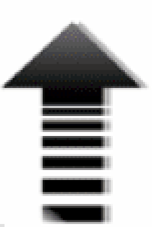
```
        // 更新第一行
        String newtotal = "total";
        for (i = 0; i <= 11; i++)
        {
            newTotaldata[i] = Integer.parseInt(beTotal[i + 1]) +
               delta[i];// 总流量数值+delta[i]更新
            newtotal = newtotal + "," + newTotaldata[i];
        }
        newtotal = newtotal + "/n";
        // 处理中间不变的部分
        String before = "";// before 为之前的从第 1 行到昨天的流量记录
        for (i = 1; i <= line.length - 2; i++)
            before = before + line[i] + "/n";  // 代表中间不变的部分
        String newlog = newtotal + before + newtoday;
        writefile(data, ONPATH);                // 更新流量记录
        writefile(newlog, LOGPATH);             // 更新 log*/
    }
}
```

13.2　基于防火墙的流量统计

在本节的内容中，将通过几段演示代码的实现过程，来讲解实现基于防火墙的 Android 流量统计的应用过程。

(1) BroadcastReceiver 模块的实现，用于监听开机信息并初始化和启动服务。具体代码如下：

```
package zy.dnh;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;
public class getpowerinfo extends BroadcastReceiver{
    FileOutputStream out;
    final public String ONPATH = "/data/data/zy.dnh/on.txt";
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
         if(intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)){
              Intent bootActivityIntent=new Intent(context,mService1.class);
                  //启动服务
              bootActivityIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
               writefile("0,0,0,0,0,0,0,0,0,0,0,0",ONPATH);
               context.startService(bootActivityIntent);
                Toast.makeText(context, "Netcounter service has been
```



```
                  lauched", Toast.LENGTH LONG).show();
               Api.applySavedIptablesRules(context, false);//应用防火墙规则
               Toast.makeText(context, "Wall rules have been lauched",
                   Toast.LENGTH LONG).show();
          }
     }
     public void writefile(String str,String path )
     {
        File file;
        try {
          //创建文件
          file = new File(path);
          file.createNewFile();
          //打开文件 file 的 OutputStream
          out = new FileOutputStream(file);
          String infoToWrite = str;
          //将字符串转换成 byte 数组写入文件
          out.write(infoToWrite.getBytes());
          //关闭文件 file 的 OutputStream
          out.close();
        } catch (IOException e) {
          //将出错信息打印到 Logcat
        }
```

(2) mService1 模块的实现，此模块是后台服务，用于维护流量日志。具体代码如下：

```
public class mService1 extends Service
{
    private Handler objHandler = new Handler();
    private int intCounter=0;
    private int mHour;
     private int mMinute;
     private int mYear;
     private int mMonth;
     private int mDay;
     private String mdate;
     final public String DEV_FILE = "/proc/self/net/dev";//系统流量文件
     String[] ethdata={"0","0","0","0","0","0","0","0","0","0","0","0",
         "0","0","0","0"};
     String[] gprsdata={"0","0","0","0","0","0","0","0","0","0","0","0",
         "0","0","0","0"};
     String[] wifidata={"0","0","0","0","0","0","0","0","0","0","0","0",
         "0","0","0","0"};
     String data="0,0,0,0,0,0,0,0,0,0,0,0"; //对应 on.txt 里面的格式
     final String ETHLINE=" eth0";          //以太网信息所在行
     final String GPRSLINE="rmnet0";
     final String WIFILINE="tiwlan0";

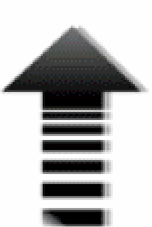
     final String TEXT ENCODING = "UTF-8";

     final public String ONPATH = "/data/data/zy.dnh/on.txt";
```

```
    final public String LOGPATH = "/data/data/zy.dnh/log.txt";
    …
```

(3) 应用 Iptable 规则模块的实现，通过运行 Iptable 脚本来实现 Iptable 规则。具体代码如下：

```
private static boolean applyIptablesRulesImpl(Context ctx, List<Integer>
uids, boolean showErrors) {
             if (ctx == null) {
                    return false;
             }
            final SharedPreferences prefs =
               ctx.getSharedPreferences(PREFS NAME, 0);
            final boolean whitelist = prefs.getString(PREF MODE,
               MODE WHITELIST).equals(MODE WHITELIST);
             boolean wifi = false; // Wi-fi selected ?
             final String itfs = prefs.getString(PREF ITFS, ITF 3G);
             String itfFilter;
             if (itfs.indexOf("|") != -1) {
                    itfFilter = ""; // Block all interfaces
                    wifi = true;
             } else if (itfs.indexOf(ITF 3G) != -1) {
                    itfFilter = "-o rmnet+";;
                    // Block all rmnet interfaces
             } else {
                    itfFilter = "-o tiwlan+";;
                    // Block all tiwlan interfaces
                    wifi = true;
             }
    final StringBuilder script = new StringBuilder();
     try {
            int code;
       script.append("iptables -F || exit/n");
        final String targetRule = (whitelist ? "ACCEPT" : "REJECT");
       if (whitelist && wifi) {
           // When "white listing" Wi-fi, we need ensure that the dhcp
              and wifi users are allowed
                    int uid = android.os.Process.getUidForName("dhcp");
          if (uid != -1) script.append("iptables -A OUTPUT "
                    + itfFilter + " -m owner --uid-owner " + uid + " -j
                      ACCEPT || exit/n");
           uid = android.os.Process.getUidForName("wifi");
          if (uid != -1) script.append("iptables -A OUTPUT " + itfFilter
             + " -m owner --uid-owner " + uid + " -j ACCEPT || exit/n"); }
        for (Integer uid : uids) {
           script.append("iptables -A OUTPUT " + itfFilter
                   + " -m owner --uid-owner " + uid + " -j " +
                      targetRule + " || exit/n");
                    }
         if (whitelist) {
             script.append("iptables -A OUTPUT " + itfFilter + " -j
```



```
                REJECT || exit/n");
                    }
            StringBuilder res = new StringBuilder();
        code = runScriptAsRoot(script.toString(), res);
        if (showErrors && code != 0) {
         String msg = res.toString();
         Log.e("DroidWall", msg);
          // Search for common error messages
         if (msg.indexOf("Couldn't find match 'owner'") != -1 ||
               msg.indexOf("no chain/target match") != -1) {
          alert(ctx, "Error applying iptables rules./nExit code: " +
                   code + "/n/n" +"It seems your Linux kernel was not
                   compiled with the netfilter /"owner/" module enabled,
                   which is required for Droid Wall to work properly./n/n"
                   + "You should check if there is an updated version of
                        your Android ROM compiled with this kernel module.");
      } else {
        // Remove unnecessary help message from output
      if (msg.indexOf("/nTry 'iptables -h' or 'iptables --help' for
             more information.") != -1) {
     msg = msg.replace("/nTry 'iptables -h' or 'iptables --help' for
              more information.", "");
    }
    // Try 'iptables -h' or 'iptables --help' for more information.
        alert(ctx, "Error applying iptables rules. Exit code: " + code +
                "/n/n" + msg.trim());
        }
     } else {
        return true;
     }
    } catch (Exception e) {
   if (showErrors) alert(ctx, "error refreshing iptables: " + e);
            }
            return false;
  }
```

13.3　适用 Android 系统的通用流量统计函数

为了便于读者使用 TrafficStats 类，笔者编写了几个流量统计函数供读者使用。这些函数保存在文件 daima\13\TrafficStatsLL.java 中，具体代码如下：

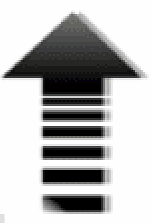
```
package com.AAJM;

import java.io.ByteArrayOutputStream;
import java.io.File;
```




```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import android.util.Log;
public class TrafficStatsLL {
/**
 * 获取网络流量信息
 * 利用读取系统文件的方法来获取网络流量
 * 主要意义在于可以应用于 2.2 以前的没有提供 TrafficStats 接口的版本
*/
    public static String readInStream(FileInputStream inStream){
          try {
           ByteArrayOutputStream outStream = new ByteArrayOutputStream();
           byte[] buffer = new byte[1024];
           int length = -1;
           while((length = inStream.read(buffer)) != -1 ){
            outStream.write(buffer, 0, length);
           }
           outStream.close();
           inStream.close();
           return outStream.toString();
          } catch (IOException e){
           Log.i("FileTest", e.getMessage());
          }
          return null;
    }
    //获取手机 2G/3G 的下载流量
    public static long getMobileRxBytes()
    {
        long ReturnLong=0;          //查询到的结果
        try {
            File file = new File("/proc/net/dev");
            FileInputStream inStream = new FileInputStream(file);
            String a=readInStream(inStream);
            int startPos=a.indexOf("rmnet0:");
            a=a.substring(startPos);
            Pattern p=Pattern.compile(" \\d+ ");
           Matcher m=p.matcher(a);
           while(m.find()){
            ReturnLong=Long.parseLong(m.group().trim());
            break;
           }

        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        }
        return ReturnLong;
    }
```



```
//获取手机 2G/3G 的上传流量
public static long getMobileTxBytes()
{
    long ReturnLong=0;              //查询到的结果
    try {
        int count=0;                //返回结果时的计数器
        File file = new File("/proc/net/dev");
        FileInputStream inStream = new FileInputStream(file);
        String a=readInStream(inStream);
        int startPos=a.indexOf("rmnet0:");
        a=a.substring(startPos);
        Pattern p=Pattern.compile(" \\d+ ");
       Matcher m=p.matcher(a);
       while(m.find()){
        if(count==8)
        {
            ReturnLong=Long.parseLong(m.group().trim());
            break;
        }
        count++;

       }

    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    }
    return ReturnLong;
}
//获取手机 Wi-Fi 的下载流量
public static long getWifiRxBytes()
{
    long ReturnLong=0;              //查询到的结果
    try {
        File file = new File("/proc/net/dev");
        FileInputStream inStream = new FileInputStream(file);
        String a=readInStream(inStream);
        int startPos=a.indexOf("wlan0:");
        a=a.substring(startPos);
        Pattern p=Pattern.compile(" \\d+ ");
       Matcher m=p.matcher(a);
       while(m.find()){
            ReturnLong=Long.parseLong(m.group().trim());
            break;
       }

    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    }
    return ReturnLong;
}
```

```
//获取手机 Wi-Fi 的上传流量/
public static long getWifiTxBytes()
{
    long ReturnLong=0;              //查询到的结果
    try {
        int count=0;                //返回结果时的计数器
        File file = new File("/proc/net/dev");
        FileInputStream inStream = new FileInputStream(file);
        String a=readInStream(inStream);
        int startPos=a.indexOf("wlan0:");
        a=a.substring(startPos);
        Pattern p=Pattern.compile(" \\d+ ");
       Matcher m=p.matcher(a);
       while(m.find()){
        if(count==8)
        {
            ReturnLong=Long.parseLong(m.group().trim());
            break;
        }
        count++;
       }

    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    }
    return ReturnLong;
}
//根据 uid 获取进程的下载流量
public static long getUidRxBytes(int uid)
{
    long ReturnLong=0;              //查询到的结果

        try {
           String  url="/proc/uid stat/"+String.valueOf(uid)+"/tcp rcv";
            File file = new File(url);
            FileInputStream inStream;
            if(file.exists())
            {
                inStream = new FileInputStream(file);
                ReturnLong=Long.parseLong(readInStream(inStream).trim());
            }
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //Log.i(url+"文件并不存在","可能原因为该文件在开机后并没有上过网，所以
            没有流量记录");
    return ReturnLong;
}
//根据 uid 获取进程的上传流量
```

```
    public static long getUidTxBytes(int uid)
    {
        long ReturnLong=0;          //查询到的结果
        try {
            String url="/proc/uid_stat/"+String.valueOf(uid)+"/tcp_snd";
            File file = new File(url);
            if(file.exists())
            {
                FileInputStream inStream = new FileInputStream(file);
                ReturnLong=Long.parseLong(readInStream(inStream).trim());
            }
        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        }
        return ReturnLong;
    }
}
```

Android

第14章 流量监控系统

流量监控是指监控当前网络内的数据流量，帮助用户及时了解系统使用了多少数据。在任何手机中，流量监控系统非常重要，特别是在上网收费的时代，及时监控自己手机的网络流量，才能在用手机上网时做到有的放矢。本章将通过一个综合流量系统实例的实现过程，讲解开发一个大型流量监控系统的具体过程。

14.1 实现流量监控功能的方式

流量监控是指对数据流进行的监控，通常包括出数据、入数据的速度和总流量。在上网时用流量监控软件可以随时获得哪些程序正在访问互联网，以及它们实时的下载速度和上传速度。在目前计算机领域中，常见的实现流量监控功能的方式有以下三种。

1. 用微软提供的API接口进行流量监控

这是很常用的一种监控方法，利用微软提供的 API 接口可以很容易地获得流量数据，但是这种方法会受多方面因素的限制，得到的结果很有可能不准确或者有延迟，甚至得不到结果。

此类方法的流量监控通常安装后即可使用而不用重启电脑。

2. 用底层驱动的方式获得网卡数据流量

这是一种不太常用的方法，技术难度较高，不易受其他因素的影响，获得的信息准确、无延迟、兼容性好，而且很容易限制被监控程序的流量。

此类方法的流量监控软件在安装后可能需要重启电脑。

3. 调用Windows中的组件获得流量数据

调用 Windows 性能工具获得网络流量，此类方法不常用、难度大，而且仅仅对 Windows 7/Vista 系统有效。基本没有软件会使用这种方法。

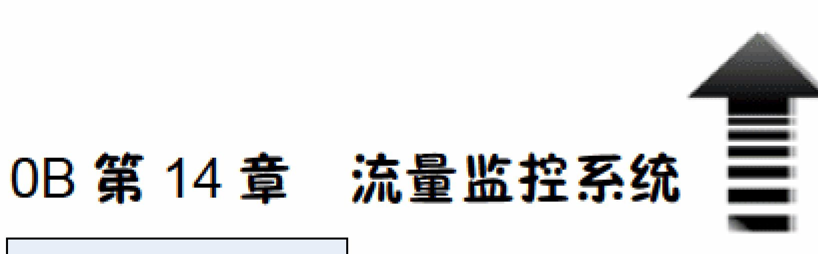
本章的网络流量防火墙系统实例采用 Android 开源系统技术，利用 Java 语言和 Eclipse 开发工具对防火墙系统进行开发。同时给出详细的系统设计流程、部分界面图及主要功能效果流程图。

实　例	功　能	源码路径
实例 15-1	开发一个网络流量防火墙	下载路径:\daima\15\wall0

在本章的内容中，还对开发过程中遇到的问题和解决方法进行详细的讨论。整个系统实例集允许上网、权限设置、系统帮助等功能于一体，在 Android 系统中能独立运行。在讲解具体编码之前，先简要介绍本项目的产生背景和项目意义，为后面的系统设计及编码做准备。

14.2 系统需求分析

根据项目的目标，我们可分析出系统的基本需求。以下从软件设计的角度来描述系统的功能，并且使用例图来描述。系统的功能模块，分别是主界面、控制界面、帮助界面、更多设置。整个系统的构成模块结构如图 14-1 所示。



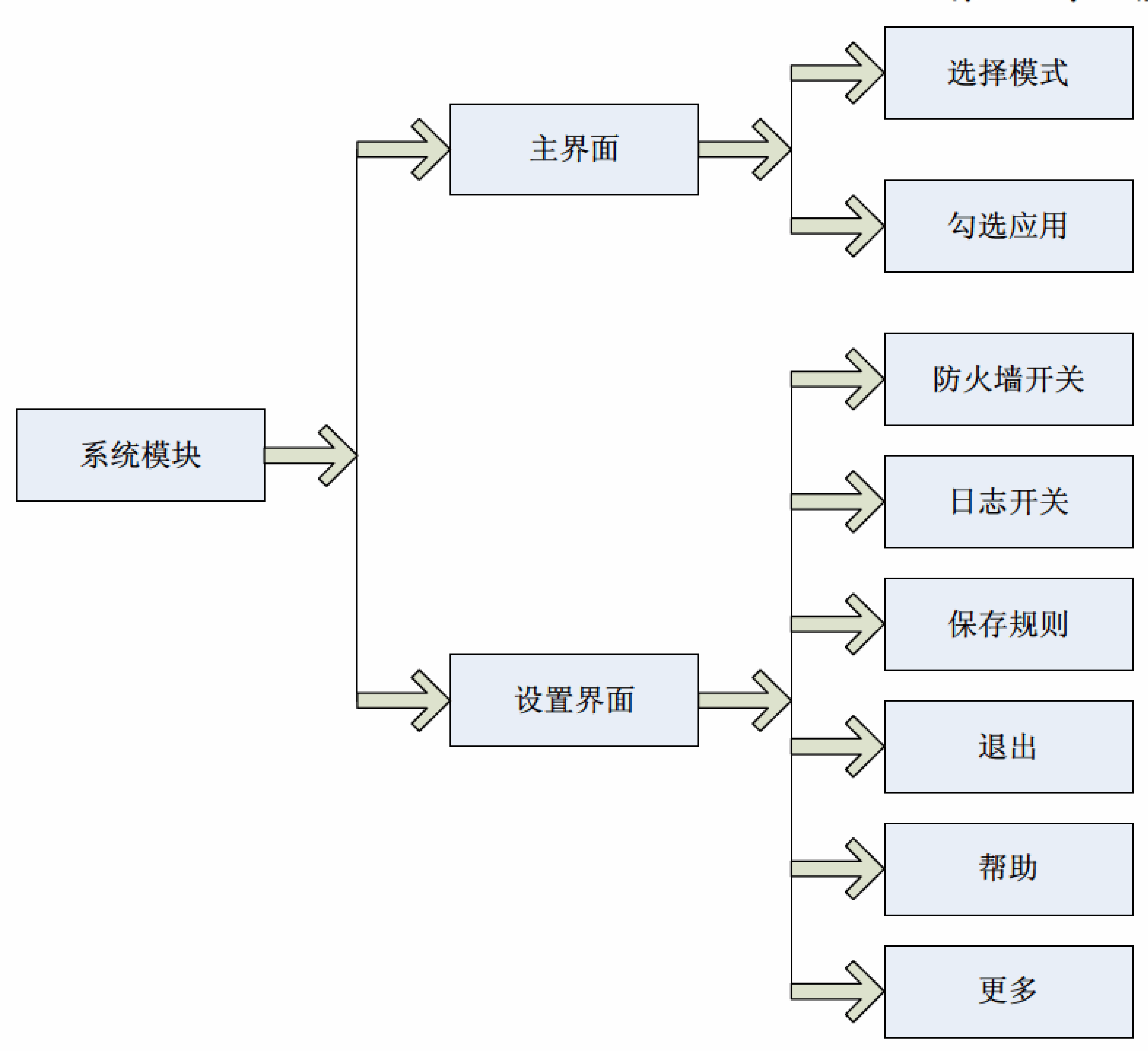


图 14-1　系统构成模块

14.3 系统需求

(1) 系统性能需求

根据 Android 手机系统要求无响应时间为 5 秒，所以就有如下性能要求。

- 邮箱类型设置：程序响应时间最长不能超过 5 秒；
- 邮箱收取设置：程序响应时间最长不能超过 5 秒；
- 邮箱发送设置：程序响应时间最长不能超过 5 秒；
- 邮箱用户检查：程序响应时间最长不能超过 5 秒；
- 用户邮件编辑：程序响应时间最长不能超过 5 秒。

(2) 运行环境需求

- 操作系统：Android 手机基于 Linux 操作系统。
- 支持环境：Android 2.3 以上版本。
- 开发环境：Eclipse 3.5 ADT 0.95。

14.4 编写布局文件

UI 布局开发是 Android 应用程序的主要工作之一。本流量监控项目包括两方面布局，

分别为主界面布局和帮助界面布局。下面将详细讲解这两部分布局文件的具体实现过程。

14.4.1 主界面布局文件main.xml

首先编写主界面文件 main.xml，系统执行之后首先显示主界面，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:layout width="fill parent"
    android:layout height="fill parent"
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:duplicateParentState="false">
    <View android:layout width="fill parent" android:layout height="1sp"
        android:background="#FFFFFFFF" />
    <LinearLayout android:layout width="fill parent"
        android:layout height="wrap content" android:padding="8sp">
        <TextView android:layout width="wrap content"
            android:layout height="wrap content" android:id=
              "@+id/label mode"
            android:text="Mode: " android:textSize="20sp" android:clickable=
              "true"></TextView>
    </LinearLayout>
    <View android:layout width="fill parent" android:layout height="1sp"
        android:background="#FFFFFFFF" />
    <RelativeLayout android:layout width="fill parent"
        android:layout height="wrap content" android:padding="3sp">
        <ImageView android:layout width="wrap content"
            android:layout height="wrap content" android:id="@+id/img wifi"
            android:src="@drawable/eth wifi" android:clickable="false"
            android:layout alignParentLeft="true" android:paddingLeft="3sp"
            android:paddingRight="10sp"></ImageView>
        <ImageView android:layout width="wrap content"
            android:layout height="wrap content" android:id="@+id/img 3g"
            android:layout toRightOf="@id/img wifi" android:src=
              "@drawable/eth g"
            android:clickable="false"></ImageView>
        <ImageView android:layout width="wrap content"
            android:layout height="wrap content" android:id="@+id/img download"
            android:src="@drawable/download"
              android:layout alignParentRight="true"
            android:paddingLeft="22sp" android:clickable="false"></ImageView>
        <ImageView android:layout width="wrap content"
            android:layout height="wrap content" android:id="@+id/img upload"
            android:layout toLeftOf="@id/img download" android:src=
              "@drawable/upload"
            android:clickable="false"></ImageView>
    </RelativeLayout>
    <ListView android:layout width="wrap content"
        android:layout height="wrap content" android:id="@+id/listview">
          </ListView>
</LinearLayout>
```



在上述代码中，将整个主界面划分为以下两个部分。

- 上部分：显示模式和网络类型，其中模式分为黑名单模式和白名单模式两种。
- 下部分：列表显示了某种模式下所有的网络服务，并且在每种服务前面显示一个复选框控件，通过该控件可以设置某种服务启用还是禁用。

下部分的列表功能是通过文件 listitem.xml 实现的，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="fill parent" android:layout height=
      "fill parent">
    <CheckBox android:layout width="wrap content"
        android:layout height="wrap content" android:id="@+id/itemcheck wifi"
        android:layout alignParentLeft="true"></CheckBox>
    <CheckBox android:layout width="wrap content"
        android:layout height="wrap content" android:id="@+id/itemcheck 3g"
        android:layout toRightOf="@id/itemcheck wifi"></CheckBox>
    <TextView android:layout height="wrap content" android:id="@+id/app text"
        android:text="uid:packages" android:layout width="match parent"
        android:layout toRightOf="@id/itemcheck 3g" android:layout
          centerVertical="true"
        android:paddingRight="80sp"></TextView>
    <TextView android:layout height="wrap content" android:id="@+id/download"
        android:layout width="wrap content" android:layout
          alignParentRight="true"
        android:layout centerVertical="true" android:paddingLeft=
          "13sp"></TextView>
    <TextView android:layout height="wrap content" android:id="@+id/upload"
        android:layout_width="wrap_content" android:layout_toLeftOf=
          "@id/download"
        android:layout centerVertical="true"></TextView>
</RelativeLayout>
```

系统主界面的效果如图 14-2 所示。

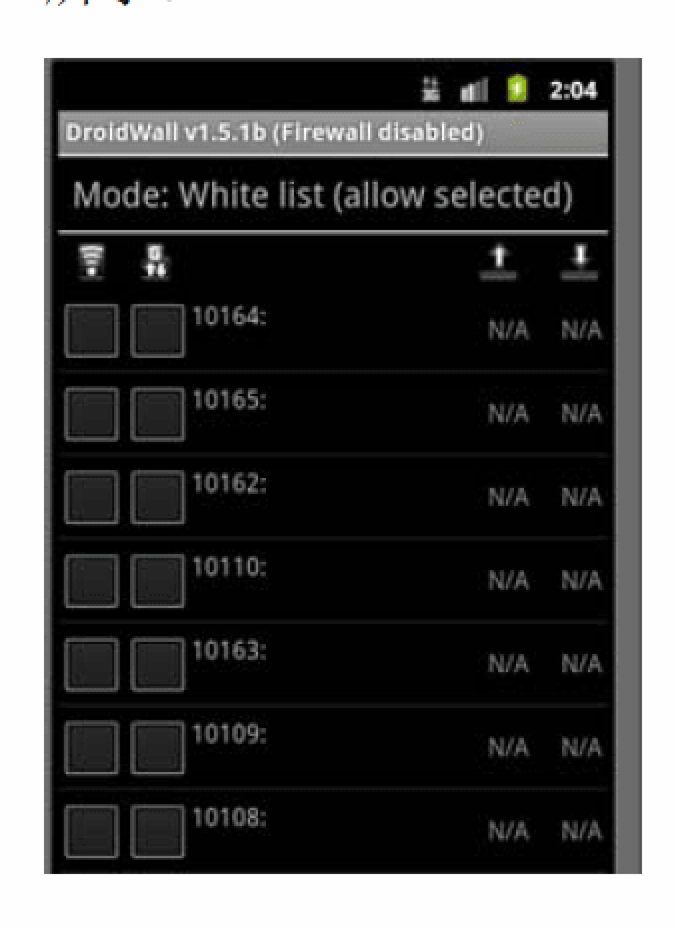


图 14-2　主界面效果

14.4.2 帮助界面布局文件help_dialog.xml

编写帮助界面布局文件 help_dialog.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="fill parent" android:layout height="wrap content">
    <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout width="fill parent" android:layout height="fill parent">
            <TextView android:layout height="fill parent"
                android:layout width="fill parent" android:text=
                  "@string/help dialog text"
                android:padding="6dip" />
    </ScrollView>
</FrameLayout>
```

系统帮助界面的效果如图 14-3 所示。



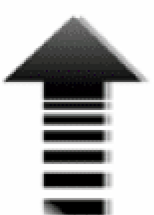
图 14-3　帮助界面效果

14.5　编写主程序文件

布局文件编写完毕之后，还需要编写值文件 strings.xml，具体代码比较简单，请读者参考本书附带光盘中的代码即可，在此不再进行详细讲解。下面将详细讲解用 Java 编写主程序文件的具体过程。

14.5.1　实现服务勾选处理和模式设置功能

首先编写文件 MainActivity.java，此文件是整个系统的核心，能够实现服务勾选处理和模式设置功能，勾选后会禁止或开启某项网络服务。文件 MainActivity.java 的具体实现流程如下。



(1) 定义类 MainActivity 为项目启动后首先显示的 Activity，设置按下 Menu 后显示的选项，并设置需要的各个实例函数。具体代码如下：

```
/**
 * 主 Activity， 当您打开应用时，这是被显示的屏幕
 */
public class MainActivity extends Activity implements
OnCheckedChangeListener,
        OnClickListener {
    // 按下 Menu 后显示的选项
    private static final int MENU DISABLE = 0;
    private static final int MENU TOGGLELOG = 1;
    private static final int MENU APPLY = 2;
    private static final int MENU EXIT = 3;
    private static final int MENU HELP = 4;
    private static final int MENU SHOWLOG = 5;
    private static final int MENU SHOWRULES = 6;
    private static final int MENU CLEARLOG = 7;
    private static final int MENU SETPWD = 8;

    /**进展对话实例*/
    private ListView listview;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        checkPreferences();
        setContentView(R.layout.main);
        this.findViewById(R.id.label mode).setOnClickListener(this);
        Api.assertBinaries(this, true);
    }

    @Override
    protected void onStart() {
        super.onStart();
        // Force re-loading the application list
        Log.d("DroidWall", "onStart() - Forcing APP list reload!");
        Api.applications = null;
    }
    @Override
    protected void onResume() {
        super.onResume();
        if (this.listview == null) {
            this.listview = (ListView) this.findViewById(R.id.listview);
        }
        refreshHeader();
        final String pwd = getSharedPreferences(Api.PREFS NAME, 0).getString(
                Api.PREF PASSWORD, "");
        if (pwd.length() == 0) {
            // No password lock
            showOrLoadApplications();
        } else {
```

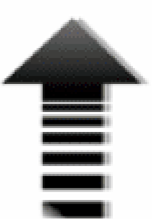
```
            // Check the password
            requestPassword(pwd);
        }
    }
    @Override
    protected void onPause() {
        super.onPause();
        this.listview.setAdapter(null);
    }
```

(2) 定义函数 checkPreferences()来检查被存储的选项是否正常，具体代码如下：

```
    /**
     * 检查被存储的选项是否正常
     */
    private void checkPreferences() {
       final SharedPreferences prefs = getSharedPreferences(Api.PREFS NAME, 0);
        final Editor editor = prefs.edit();
        boolean changed = false;
        if (prefs.getString(Api.PREF MODE, "").length() == 0) {
            editor.putString(Api.PREF MODE, Api.MODE WHITELIST);
            changed = true;
        }
        /* 删除旧的选项名字 */
        if (prefs.contains("AllowedUids")) {
            editor.remove("AllowedUids");
            changed = true;
        }
        if (prefs.contains("Interfaces")) {
            editor.remove("Interfaces");
            changed = true;
        }
        if (changed)
            editor.commit();
    }
```

(3) 定义函数 refreshHeader()来刷新显示当前运行的和网络相关的程序，具体代码如下：

```
    /**
     * 刷新显示当前运行的和网络相关的程序
     */
    private void refreshHeader() {
        final SharedPreferences prefs = getSharedPreferences(Api.PREFS NAME, 0);
       final String mode = prefs.getString(Api.PREF MODE, Api.MODE WHITELIST);
        final TextView labelmode = (TextView) this
            .findViewById(R.id.label mode);
        final Resources res = getResources();
        int resid = (mode.equals(Api.MODE WHITELIST) ? R.string.mode whitelist
            : R.string.mode blacklist);
        labelmode.setText(res.getString(R.string.mode header,
            res.getString(resid)));
        resid = (Api.isEnabled(this) ? R.string.title_enabled
```



```
            : R.string.title disabled);
        setTitle(res.getString(resid, Api.VERSION));
    }
```

(4) 定义函数 selectMode()显示对话框选择操作方式，供我们选择黑名单模式还是白名单模式。具体代码如下：

```
    /**
     * 显示对话框选择操作方式，供我们选择黑名单模式还是白名单模式
     */
    private void selectMode() {
        final Resources res = getResources();
        new AlertDialog.Builder(this)
                .setItems(
                    new String[] { res.getString(R.string.mode whitelist),
                    res.getString(R.string.mode blacklist) },
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
                            int which) {
                        final String mode = (which == 0 ? Api.MODE WHITELIST
                            : Api.MODE BLACKLIST);
                        final Editor editor = getSharedPreferences(
                            Api.PREFS NAME, 0).edit();
                        editor.putString(Api.PREF MODE, mode);
                            editor.commit();
                            refreshHeader();
                        }
                    }).setTitle("Select mode:").show();
    }
```

(5) 定义函数 setPassword()来设置一个系统密码，如果设置密码后，在进入主界面前会通过函数 requestPassword()来验证密码，只有密码正确才能进入。具体代码如下：

```
    /**
     * 设置一新的密码
     */
    private void setPassword(String pwd) {
        final Resources res = getResources();
        final Editor editor = getSharedPreferences(Api.PREFS NAME, 0).edit();
        editor.putString(Api.PREF PASSWORD, pwd);
        String msg;
        if (editor.commit()) {
            if (pwd.length() > 0) {
                msg = res.getString(R.string.passdefined);
            } else {
                msg = res.getString(R.string.passremoved);
            }
        } else {
            msg = res.getString(R.string.passerror);
        }
        Toast.makeText(MainActivity.this, msg, Toast.LENGTH SHORT).show();
    }
```



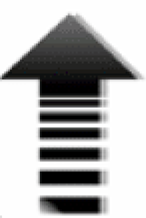

```
    /**
     * 如果设置了密码，显示主界面前先验证密码
     */
    private void requestPassword(final String pwd) {
        new PassDialog(this, false, new android.os.Handler.Callback() {
            public boolean handleMessage(Message msg) {
                if (msg.obj == null) {
                    MainActivity.this.finish();
                    android.os.Process.killProcess(android.os.Process.myPid());
                    return false;
                }
                if (!pwd.equals(msg.obj)) {
                    requestPassword(pwd);
                    return false;
                }
                // 如果密码正确
                showOrLoadApplications();
                return false;
            }
        }).show();
    }
```

(6) 编写函数 toggleLogEnabled()来实现防火墙禁用和日志禁用开关处理，具体代码如下：

```
    /**
     * 开关设置
     */
    private void toggleLogEnabled() {
        final SharedPreferences prefs = getSharedPreferences(Api.PREFS NAME, 0);
        final boolean enabled = !prefs.getBoolean(Api.PREF LOGENABLED, false);
        final Editor editor = prefs.edit();
        editor.putBoolean(Api.PREF LOGENABLED, enabled);
        editor.commit();
        if (Api.isEnabled(this)) {
            Api.applySavedIptablesRules(this, true);
        }
        Toast.makeText(
                MainActivity.this,
                (enabled ? R.string.log was enabled : R.string.log was disabled),
                Toast.LENGTH SHORT).show();
    }
```

(7) 编写函数 showOrLoadApplications()，如果在某模式下有应用则显示其中的应用。函数 showOrLoadApplications()的具体代码如下：

```
    /**
     * 如果某模式下有应用，则显示其中的应用
     */
    private void showOrLoadApplications() {
        final Resources res = getResources();
```



```
        if (Api.applications == null) {
            final ProgressDialog progress = ProgressDialog.show(this,
                    res.getString(R.string.working),
                    res.getString(R.string.reading apps), true);
            final Handler handler = new Handler() {
                public void handleMessage(Message msg) {
                    try {
                        progress.dismiss();
                    } catch (Exception ex) {
                    }
                    showApplications();
                }
            };
            new Thread() {
                public void run() {
                    Api.getApps(MainActivity.this);
                    handler.sendEmptyMessage(0);
                }
            }.start();
        } else {
            // 储存应用，显示名单
            showApplications();
        }
    }
```

(8) 编写函数 showApplications()显示应用名单，具体代码如下：

```
    /**
     * 显示应用名单
     */
    private void showApplications() {
        final DroidApp[] apps = Api.getApps(this);
        // Sort applications - selected first, then alphabetically
        Arrays.sort(apps, new Comparator<DroidApp>() {
            @Override
            public int compare(DroidApp o1, DroidApp o2) {
                if ((o1.selected wifi | o1.selected 3g) ==
                    (o2.selected wifi | o2.selected 3g)) {
                    return String.CASE INSENSITIVE ORDER.compare(o1.names[0],
                            o2.names[0]);
                }
                if (o1.selected wifi || o1.selected 3g)
                    return -1;
                return 1;
            }
        });
        final LayoutInflater inflater = getLayoutInflater();
        final ListAdapter adapter = new ArrayAdapter<DroidApp>(this,
                R.layout.listitem, R.id.app text, apps) {
            @Override
            public View getView(int position, View convertView, ViewGroup
```



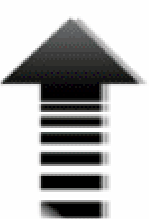

```
                parent) {
                ListEntry entry;
                if (convertView == null) {
                    // Inflate a new view
                   convertView = inflater.inflate(R.layout.listitem, parent,
                            false);
                    entry = new ListEntry();
                    entry.box wifi = (CheckBox) convertView
                            .findViewById(R.id.itemcheck wifi);
                    entry.box 3g = (CheckBox) convertView
                            .findViewById(R.id.itemcheck 3g);
                    entry.app text = (TextView) convertView
                            .findViewById(R.id.app text);
                    entry.upload = (TextView) convertView
                            .findViewById(R.id.upload);
                    entry.download = (TextView) convertView
                            .findViewById(R.id.download);
                    convertView.setTag(entry);
                    entry.box wifi

    .setOnCheckedChangeListener(MainActivity.this);

    entry.box 3g.setOnCheckedChangeListener(MainActivity.this);
                } else {
                    //转换一个现有视图
                    entry = (ListEntry) convertView.getTag();
                }
                final DroidApp app = apps[position];
                entry.app text.setText(app.toString());
                convertAndSetColor(TrafficStats.getUidTxBytes(app.uid),
                    entry.upload);
                convertAndSetColor(TrafficStats.getUidRxBytes(app.uid),
                    entry.download);
                final CheckBox box wifi = entry.box wifi;
                box wifi.setTag(app);
                box wifi.setChecked(app.selected wifi);
                final CheckBox box 3g = entry.box 3g;
                box 3g.setTag(app);
                box 3g.setChecked(app.selected 3g);
                return convertView;
            }
```

(9) 编写函数 convertAndSetColor()，根据对某选项的设置显示内容，并设置显示内容的颜色。假如没有任何设置，则显示“N/A”，如果已经设置了启用，则显示已经用过的流量。函数 convertAndSetColor()的具体代码如下：

```
        private void convertAndSetColor(long num, TextView text) {
            String value = null;
            long temp = num;
            float floatnum = num;
            if (num == -1) {
```



```
                    value = "N/A ";
                    text.setText(value);
                    text.setTextColor(0xff919191);
                    return ;
                } else if ((temp = temp / 1024) < 1) {
                    value = num + "B";
                } else if ((floatnum = temp / 1024) < 1) {
                    value = temp + "KB";
                } else {
                    DecimalFormat format = new DecimalFormat("##0.0");
                    value = format.format(floatnum) + "MB";
                }
                text.setText(value);
                text.setTextColor(0xffff0300);
            }
        };
        this.listview.setAdapter(adapter);
}
```

(10) 进入系统主界面后，如果按下 Menu 键则会弹出设置界面，在设置界面中可以选择对应的功能。在设置界面中的选择功能是通过以下三个函数实现的。

```
public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, MENU DISABLE, 0, R.string.fw enabled).setIcon(
                android.R.drawable.button onoff indicator on);
        menu.add(0, MENU TOGGLELOG, 0, R.string.log enabled).setIcon(
                android.R.drawable.button onoff indicator on);
        menu.add(0, MENU APPLY, 0, R.string.applyrules).setIcon(
                R.drawable.apply);
        menu.add(0, MENU EXIT, 0, R.string.exit).setIcon(
                android.R.drawable.ic menu close clear cancel);
        menu.add(0, MENU HELP, 0, R.string.help).setIcon(
                android.R.drawable.ic_menu_help);
        menu.add(0, MENU SHOWLOG, 0, R.string.show log)
                .setIcon(R.drawable.show);
        menu.add(0, MENU SHOWRULES, 0, R.string.showrules).setIcon(
                R.drawable.show);
        menu.add(0, MENU CLEARLOG, 0, R.string.clear log).setIcon(
                android.R.drawable.ic menu close clear cancel);
        menu.add(0, MENU SETPWD, 0, R.string.setpwd).setIcon(
                android.R.drawable.ic lock lock);
        return true;
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        final MenuItem item onoff = menu.getItem(MENU DISABLE);
        final MenuItem item apply = menu.getItem(MENU APPLY);
        final boolean enabled = Api.isEnabled(this);
        if (enabled) {
```

```
    item_onoff.setIcon(android.R.drawable.button_onoff_indicator_on);
            item_onoff.setTitle(R.string.fw_enabled);
            item_apply.setTitle(R.string.applyrules);
        } else {
            item_onoff.setIcon(android.R.drawable.button_onoff_indicator_off);
            item_onoff.setTitle(R.string.fw_disabled);
            item_apply.setTitle(R.string.saverules);
        }
        final MenuItem item_log = menu.getItem(MENU_TOGGLELOG);
        final boolean logenabled = getSharedPreferences(Api.PREFS_NAME, 0)
                .getBoolean(Api.PREF_LOGENABLED, false);
        if (logenabled) {
            item_log.setIcon(android.R.drawable.button_onoff_indicator_on);
            item_log.setTitle(R.string.log_enabled);
        } else {
            item_log.setIcon(android.R.drawable.button_onoff_indicator_off);
            item_log.setTitle(R.string.log_disabled);
        }
        return super.onPrepareOptionsMenu(menu);
    }

    @Override
    public boolean onMenuItemSelected(int featureId, MenuItem item) {
        switch (item.getItemId()) {
        case MENU_DISABLE:
            disableOrEnable();
            return true;
        case MENU_TOGGLELOG:
            toggleLogEnabled();
            return true;
        case MENU_APPLY:
            applyOrSaveRules();
            return true;
        case MENU_EXIT:
            finish();
            System.exit(0);
            return true;
        case MENU_HELP:
            new HelpDialog(this).show();
            return true;
        case MENU_SETPWD:
            setPassword();
            return true;
        case MENU_SHOWLOG:
            showLog();
            return true;
        case MENU_SHOWRULES:
            showRules();
            return true;
        case MENU_CLEARLOG:
            clearLog();
```

```
            return true;
        }
        return false;
    }
```

(11) 编写函数 disableOrEnable()设置开启或关闭防火墙，具体代码如下：

```
    private void disableOrEnable() {
        final boolean enabled = !Api.isEnabled(this);
        Log.d("DroidWall", "Changing enabled status to: " + enabled);
        Api.setEnabled(this, enabled);
        if (enabled) {
            applyOrSaveRules();
        } else {
            purgeRules();
        }
        refreshHeader();
    }
```

(12) 编写函数 setPassword()来到设置密码界面，具体代码如下：

```
    private void setPassword() {
        new PassDialog(this, true, new android.os.Handler.Callback() {
            public boolean handleMessage(Message msg) {
                if (msg.obj != null) {
                    setPassword((String) msg.obj);
                }
                return false;
            }
        }).show();
    }
```

(13) 选择 Save rules(保存规则)后执行函数 showRules()，具体代码如下：

```
    private void showRules() {
        final Resources res = getResources();
        final ProgressDialog progress = ProgressDialog.show(this,
                res.getString(R.string.working),
                res.getString(R.string.please wait), true);
        final Handler handler = new Handler() {
            public void handleMessage(Message msg) {
                try {
                    progress.dismiss();
                } catch (Exception ex) {
                }
                if (!Api.hasRootAccess(MainActivity.this, true))
                    return;
                Api.showIptablesRules(MainActivity.this);
            }
        };
        handler.sendEmptyMessageDelayed(0, 100);
    }
```

(14) 编写函数 showLog()显示日志信息界面，具体代码如下：

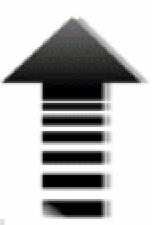
```
private void showLog() {
    final Resources res = getResources();
    final ProgressDialog progress = ProgressDialog.show(this,
            res.getString(R.string.working),
            res.getString(R.string.please wait), true);
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            try {
                progress.dismiss();
            } catch (Exception ex) {
            }
            Api.showLog(MainActivity.this);
        }
    };
    handler.sendEmptyMessageDelayed(0, 100);
}
```

(15) 编写函数 clearLog()清除系统内的日志记录信息，具体代码如下：

```
private void clearLog() {
    final Resources res = getResources();
    final ProgressDialog progress = ProgressDialog.show(this,
            res.getString(R.string.working),
            res.getString(R.string.please wait), true);
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            try {
                progress.dismiss();
            } catch (Exception ex) {
            }
            if (!Api.hasRootAccess(MainActivity.this, true))
                return;
            if (Api.clearLog(MainActivity.this)) {
                Toast.makeText(MainActivity.this, R.string.log cleared,
                Toast.LENGTH SHORT).show();
            }
        }
    };
    handler.sendEmptyMessageDelayed(0, 100);
}
```

(16) 编写函数 applyOrSaveRules()，当申请或保存规则后将规则运用到本系统。具体代码如下：

```
private void applyOrSaveRules() {
    final Resources res = getResources();
    final boolean enabled = Api.isEnabled(this);
    final ProgressDialog progress = ProgressDialog.show(this, res
            .getString(R.string.working), res
            .getString(enabled ? R.string.applying_rules
```



```
                    : R.string.saving rules), true);
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            try {
                progress.dismiss();
            } catch (Exception ex) {
            }
            if (enabled) {
                Log.d("DroidWall", "Applying rules.");
                if (Api.hasRootAccess(MainActivity.this, true)
                    && Api.applyIptablesRules(MainActivity.this, true)) {
                    Toast.makeText(MainActivity.this,
                        R.string.rules applied, Toast.LENGTH SHORT)
                        .show();
                } else {
                    Log.d("DroidWall", "Failed - Disabling firewall.");
                    Api.setEnabled(MainActivity.this, false);
                }
            } else {
                Log.d("DroidWall", "Saving rules.");
                Api.saveRules(MainActivity.this);
                Toast.makeText(MainActivity.this, R.string.rules saved,
                Toast.LENGTH SHORT).show();
            }
        }
    };
    handler.sendEmptyMessageDelayed(0, 100);
}
```

(17) 编写函数 purgeRules()来清除一个规则，具体代码如下：

```
private void purgeRules() {
    final Resources res = getResources();
    final ProgressDialog progress = ProgressDialog.show(this,
            res.getString(R.string.working),
            res.getString(R.string.deleting rules), true);
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            try {
                progress.dismiss();
            } catch (Exception ex) {
            }
            if (!Api.hasRootAccess(MainActivity.this, true))
                return;
            if (Api.purgeIptables(MainActivity.this, true)) {
                Toast.makeText(MainActivity.this, R.string.rules deleted,
                Toast.LENGTH SHORT).show();
            }
        }
    };
    handler.sendEmptyMessageDelayed(0, 100);
}
```

(18) 编写函数 onCheckedChanged()检查 Wi-Fi 选项和 3G 选项是否发生变化。具体代码如下：

```
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        final DroidApp app = (DroidApp) buttonView.getTag();
        if (app != null) {
            switch (buttonView.getId()) {
            case R.id.itemcheck_wifi:
                app.selected_wifi = isChecked;
                break;
            case R.id.itemcheck_3g:
                app.selected_3g = isChecked;
                break;
            }
        }
    }
```

到此为止，主界面程序介绍完毕，按下 Menu 键后会弹出设置界面，如图 14-4 所示。

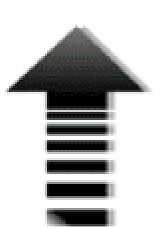


图 14-4　设置界面效果

14.5.2　实现帮助模块

编写文件 HelpDialog.java。单击设置界面中的 ? 按钮将会弹出帮助界面。文件 HelpDialog.java 的具体代码如下：

```
import android.app.AlertDialog;
import android.content.Context;
import android.view.View;
public class HelpDialog extends AlertDialog {
    protected HelpDialog(Context context) {
        super(context);
        final View view = getLayoutInflater().inflate(R.layout.
            help_dialog, null);
        setButton(context.getText(R.string.close), (OnClickListener)null);
        setIcon(R.drawable.icon);
        setTitle("DroidWall v" + Api.VERSION);
```



```
        setView(view);
    }
}
```

14.5.3　实现公共库函数

编写文件 Api.java，在此文件中定义项目中需要的公共库函数。为了便于项目的开发，专门用此文件保存了系统中经常需要的函数。文件 Api.java 的具体实现流程如下。

(1) 编写函数 scriptHeader()创建一个通用的 Script 程序头，此程序可供二进制数据使用。具体代码如下：

```
    private static String scriptHeader(Context ctx) {
        final String dir = ctx.getDir("bin", 0).getAbsolutePath();
        final String myiptables = dir + "/iptables armv5";
        return "" + "IPTABLES=iptables\n" + "BUSYBOX=busybox\n" + "GREP=grep\n"
                + "ECHO=echo\n" + "# Try to find busybox\n" + "if "
                + dir
                + "/busybox g1 --help >/dev/null 2>/dev/null ; then\n"
                + " BUSYBOX="
                + dir
                + "/busybox g1\n"
                + " GREP=\"$BUSYBOX grep\"\n"
                + " ECHO=\"$BUSYBOX echo\"\n"
                + "elif busybox --help >/dev/null 2>/dev/null ; then\n"
                + " BUSYBOX=busybox\n"
                + "elif /system/xbin/busybox --help >/dev/null
                  2>/dev/null ; then\n"
                + " BUSYBOX=/system/xbin/busybox\n"
                + "elif /system/bin/busybox --help >/dev/null
                  2>/dev/null ; then\n"
                + " BUSYBOX=/system/bin/busybox\n"
                + "fi\n"
                + "# Try to find grep\n"
                + "if ! $ECHO 1 | $GREP -q 1 >/dev/null 2>/dev/null ; then\n"
                + " if $ECHO 1 | $BUSYBOX grep -q 1 >/dev/null
                  2>/dev/null ; then\n"
                + "     GREP=\"$BUSYBOX grep\"\n"
                + " fi\n"
                + " # Grep is absolutely required\n"
                + " if ! $ECHO 1 | $GREP -q 1 >/dev/null 2>/dev/null ; then\n"
                + "     $ECHO The grep command is required. DroidWall
                  will not work.\n"
                + "     exit 1\n"
                + " fi\n"
                + "fi\n"
                + "# Try to find iptables\n"
                + "if "
                + myiptables
                + " --version >/dev/null 2>/dev/null ; then\n"
```

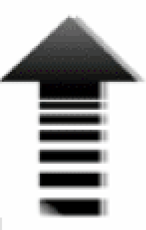
```
                + " IPTABLES="
                + myiptables + "\n" + "fi\n" + "";
    }
```

(2) 编写函数 copyRawFile()，复制一个未加工的资源文件，根据其 ID 给特定地点。具体代码如下：

```
    private static void copyRawFile(Context ctx, int resid, File file,
            String mode) throws IOException, InterruptedException {
        final String abspath = file.getAbsolutePath();
        // 在 Iptables 写入二进制数据
        final FileOutputStream out = new FileOutputStream(file);
       final InputStream is = ctx.getResources().openRawResource(resid);
        byte buf[] = new byte[1024];
        int len;
        while ((len = is.read(buf)) > 0) {
            out.write(buf, 0, len);
        }
        out.close();
        is.close();
        // 允许改变
        Runtime.getRuntime().exec("chmod " + mode + " " + abspath).waitFor();
    }
```

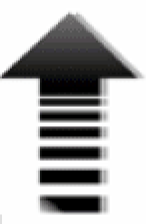
(3) 编写函数 applyIptablesRulesImpl()，功能是清洗并且重新加写所有规则，此功能是在内部实施的。函数 applyIptablesRulesImpl()的具体代码如下：

```
    private static boolean applyIptablesRulesImpl(Context ctx,
            List<Integer> uidsWifi, List<Integer> uids3g, boolean showErrors) {
        if (ctx == null) {
            return false;
        }
        assertBinaries(ctx, showErrors);
        final String ITFS WIFI[] = { "tiwlan+", "wlan+", "eth+" };
        final String ITFS 3G[] = { "rmnet+", "pdp+", "ppp+", "uwbr+", "wimax+",
                "vsnet+" };
        final SharedPreferences prefs = ctx.getSharedPreferences(PREFS NAME, 0);
        final boolean whitelist = prefs.getString(PREF MODE, MODE WHITELIST)
                .equals(MODE WHITELIST);
        final boolean blacklist = !whitelist;
        final boolean logenabled = ctx.getSharedPreferences(PREFS NAME, 0)
                .getBoolean(PREF LOGENABLED, false);
        final StringBuilder script = new StringBuilder();
        try {
            int code;
            script.append(scriptHeader(ctx));
            script.append(""
                    + "$IPTABLES --version || exit 1\n"
                    + "# Create the droidwall chains if necessary\n"
                    + "$IPTABLES -L droidwall >/dev/null 2>/dev/null ||
                      $IPTABLES --new droidwall || exit 2\n"
                    + "$IPTABLES -L droidwall-3g >/dev/null 2>/dev/null
```



```
                || $IPTABLES --new droidwall-3g || exit 3\n"
            + "$IPTABLES -L droidwall-wifi >/dev/null 2>/dev/null
                || $IPTABLES --new droidwall-wifi || exit 4\n"
            + "$IPTABLES -L droidwall-reject >/dev/null 2>/dev/null
                || $IPTABLES --new droidwall-reject || exit 5\n"
            + "# Add droidwall chain to OUTPUT chain if necessary\n"
            + "$IPTABLES -L OUTPUT | $GREP -q droidwall
                || $IPTABLES -A OUTPUT -j droidwall || exit 6\n"
            + "# Flush existing rules\n"
            + "$IPTABLES -F droidwall || exit 7\n"
            + "$IPTABLES -F droidwall-3g || exit 8\n"
            + "$IPTABLES -F droidwall-wifi || exit 9\n"
            + "$IPTABLES -F droidwall-reject || exit 10\n" + "");
// 检查是否能设置
if (logenabled) {
    script.append(""
                + "# Create the log and reject rules (ignore
                    errors on the LOG target just in case it is
                    not available)\n"
                + "$IPTABLES -A droidwall-reject -j LOG --log-
                    prefix \"[DROIDWALL] \" --log-uid\n"
                + "$IPTABLES -A droidwall-reject -j REJECT
                    || exit 11\n"
                + "");
} else {
    script.append(""
                + "# Create the reject rule (log disabled)\n"
                + "$IPTABLES -A droidwall-reject -j REJECT
                    || exit 11\n"
                + "");
}
if (whitelist && logenabled) {
    script.append("# Allow DNS lookups on white-list for a
        better logging (ignore errors)\n");
    script.append("$IPTABLES -A droidwall -p udp --dport 53
        -j RETURN\n");
}
script.append("# Main rules (per interface)\n");
for (final String itf : ITFS_3G) {
    script.append("$IPTABLES -A droidwall -o ").append(itf)
            .append(" -j droidwall-3g || exit\n");
}
for (final String itf : ITFS_WIFI) {
    script.append("$IPTABLES -A droidwall -o ").append(itf)
            .append(" -j droidwall-wifi || exit\n");
}
script.append("# Filtering rules\n");
final String targetRule = (whitelist ? "RETURN"
        : "droidwall-reject");
final boolean any_3g = uids3g.indexOf(SPECIAL_UID_ANY) >= 0;
final boolean any_wifi = uidsWifi.indexOf(SPECIAL_UID_ANY) >= 0;
```

```
if (whitelist && !any wifi) {
    //当设置开启 WI-FI 时需要保证用户允许 DHCP 和 WI-FI 功能
    int uid = android.os.Process.getUidForName("dhcp");
    if (uid != -1) {
        script.append("# dhcp user\n");
        script.append(
            "$IPTABLES -A droidwall-wifi -m owner --uid-owner ")
            .append(uid).append(" -j RETURN || exit\n");
    }
    uid = android.os.Process.getUidForName("wifi");
    if (uid != -1) {
        script.append("# wifi user\n");
        script.append(
            "$IPTABLES -A droidwall-wifi -m owner --uid-owner ")
            .append(uid).append(" -j RETURN || exit\n");
    }
}
if (any 3g) {
    if (blacklist) {
        /* block any application on this interface */
        script.append("$IPTABLES -A droidwall-3g -j ")
                .append(targetRule).append(" || exit\n");
    }
} else {
    /*释放或阻塞在这个接口的各自的应用*/
    for (final Integer uid : uids3g) {
        if (uid >= 0)
            script.append(
                "$IPTABLES -A droidwall-3g -m owner --uid-owner ")
                .append(uid).append(" -j ").append(targetRule)
                .append(" || exit\n");
    }
}
if (any wifi) {
    if (blacklist) {
        /*阻塞在这个接口的所有应用*/
        script.append("$IPTABLES -A droidwall-wifi -j ")
                .append(targetRule).append(" || exit\n");
    }
} else {
    /*释放或阻塞在这个接口的各自的应用*/
    for (final Integer uid : uidsWifi) {
        if (uid >= 0)
            script.append(
               "$IPTABLES -A droidwall-wifi -m owner --uid-owner ")
               .append(uid).append(" -j ").append(targetRule)
                .append(" || exit\n");
    }
}
if (whitelist) {
    if (!any_3g) {
```

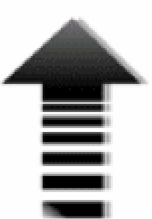


```
            if (uids3g.indexOf(SPECIAL UID KERNEL) >= 0) {
                script.append("# hack to allow kernel packets on
                    white-list\n");
                script.append("$IPTABLES -A droidwall-3g -m owner
                    --uid-owner 0:999999999 -j droidwall-reject
                    || exit\n");
            } else {
                script.append("$IPTABLES -A droidwall-3g -j
                    droidwall-reject || exit\n");
            }
        }
        if (!any wifi) {
            if (uidsWifi.indexOf(SPECIAL UID KERNEL) >= 0) {
                script.append("# hack to allow kernel packets on
                    white-list\n");
                script.append("$IPTABLES -A droidwall-wifi -m
                    owner --uid-owner 0:999999999 -j droidwall-
                    reject || exit\n");
            } else {
                script.append("$IPTABLES -A droidwall-wifi -j
                    droidwall-reject || exit\n");
            }
        }
    } else {
        if (uids3g.indexOf(SPECIAL UID KERNEL) >= 0) {
            script.append("# hack to BLOCK kernel packets on
                black-list\n");
            script.append("$IPTABLES -A droidwall-3g -m owner --
                uid-owner 0:999999999 -j RETURN || exit\n");
            script.append("$IPTABLES -A droidwall-3g -j
                droidwall-reject || exit\n");
        }
        if (uidsWifi.indexOf(SPECIAL UID KERNEL) >= 0) {
            script.append("# hack to BLOCK kernel packets on
                black-list\n");
            script.append("$IPTABLES -A droidwall-wifi -m owner -
                -uid-owner 0:999999999 -j RETURN || exit\n");
            script.append("$IPTABLES -A droidwall-wifi -j
                droidwall-reject || exit\n");
        }
    }
    final StringBuilder res = new StringBuilder();
    code = runScriptAsRoot(ctx, script.toString(), res);
    if (showErrors && code != 0) {
        String msg = res.toString();
        Log.e("DroidWall", msg);
        // 去除多余的帮助信息
        if (msg.indexOf("\nTry 'iptables -h' or 'iptables --help'
            for more information.") != -1) {
            msg = msg
                .replace(
```

```
                "\nTry 'iptables -h' or 'iptables --help' for
                    more information.",
                "");
            }
            alert(ctx, "Error applying iptables rules. Exit code: " + code
                + "\n\n" + msg.trim());
        } else {
            return true;
        }
    } catch (Exception e) {
        if (showErrors)
            alert(ctx, "error refreshing iptables: " + e);
    }
    return false;
}
```

(4) 编写函数 applySavedIptablesRules()，功能是清洗并且重新加写所有规则，此规则不是内存中保存的。因为不需要读安装引用程序，所以此方法比函数 applyIptablesRulesImpl()方式快。函数 applySavedIptablesRules()的具体代码如下：

```
public static boolean applySavedIptablesRules(Context ctx,
        boolean showErrors) {
    if (ctx == null) {
        return false;
    }
    final SharedPreferences prefs = ctx.getSharedPreferences(PREFS_NAME, 0);
    final String savedUids_wifi = prefs.getString(PREF_WIFI_UIDS, "");
    final String savedUids_3g = prefs.getString(PREF_3G_UIDS, "");
    final List<Integer> uids_wifi = new LinkedList<Integer>();
    if (savedUids_wifi.length() > 0) {
        // 检查哪一些应用使用 Wi-Fi
        final StringTokenizer tok = new StringTokenizer(savedUids_wifi, "|");
        while (tok.hasMoreTokens()) {
            final String uid = tok.nextToken();
            if (!uid.equals("")) {
                try {
                    uids_wifi.add(Integer.parseInt(uid));
                } catch (Exception ex) {
                }
            }
        }
    }
    final List<Integer> uids_3g = new LinkedList<Integer>();
    if (savedUids_3g.length() > 0) {
        //检查哪些应用允许 2G/3G 服务
        final StringTokenizer tok = new StringTokenizer(savedUids_3g, "|");
        while (tok.hasMoreTokens()) {
            final String uid = tok.nextToken();
            if (!uid.equals("")) {
                try {
                    uids_3g.add(Integer.parseInt(uid));
```



```
                    } catch (Exception ex) {
                    }
                }
            }
        }
        return applyIptablesRulesImpl(ctx, uids wifi, uids 3g, showErrors);
    }
```

(5) 编写函数 saveRules()，根据设置的选择项保存当前的规则，具体代码如下：

```
    public static void saveRules(Context ctx) {
        final SharedPreferences prefs = ctx.getSharedPreferences(PREFS NAME, 0);
        final DroidApp[] apps = getApps(ctx);
        // 建立被隔离的名单列表
        final StringBuilder newuids wifi = new StringBuilder();
        final StringBuilder newuids 3g = new StringBuilder();
        for (int i = 0; i < apps.length; i++) {
            if (apps[i].selected wifi) {
                if (newuids wifi.length() != 0)
                    newuids wifi.append('|');
                newuids wifi.append(apps[i].uid);
            }
            if (apps[i].selected 3g) {
                if (newuids 3g.length() != 0)
                    newuids 3g.append('|');
                newuids_3g.append(apps[i].uid);
            }
        }
        //除 UIDs 新的名单之外
        final Editor edit = prefs.edit();
        edit.putString(PREF WIFI UIDS, newuids wifi.toString());
        edit.putString(PREF 3G UIDS, newuids 3g.toString());
        edit.commit();
    }
```

(6) 编写函数 purgeIptables()清除所有的过滤规则，具体代码如下：

```
    public static boolean purgeIptables(Context ctx, boolean showErrors) {
        StringBuilder res = new StringBuilder();
        try {
            assertBinaries(ctx, showErrors);
            int code = runScriptAsRoot(ctx, scriptHeader(ctx)
                    + "$IPTABLES -F droidwall\n"
                    + "$IPTABLES -F droidwall-reject\n"
                    + "$IPTABLES -F droidwall-3g\n"
                    + "$IPTABLES -F droidwall-wifi\n", res);
            if (code == -1) {
                if (showErrors)
                    alert(ctx, "error purging iptables. exit code: " + code
                            + "\n" + res);
                return false;
            }
```

```
            return true;
        } catch (Exception e) {
            if (showErrors)
                alert(ctx, "error purging iptables: " + e);
            return false;
        }
    }
```

(7) 编写函数 clearLog()清除系统中的日志记录信息，具体代码如下：

```
    public static boolean clearLog(Context ctx) {
        try {
            final StringBuilder res = new StringBuilder();
            int code = runScriptAsRoot(ctx, "dmesg -c >/dev/null || exit\n",
                    res);
            if (code != 0) {
                alert(ctx, res);
                return false;
            }
            return true;
        } catch (Exception e) {
            alert(ctx, "error: " + e);
        }
        return false;
    }
```

(8) 编写函数 showLog()显示系统中的日志记录信息，具体代码如下：

```
    public static void showLog(Context ctx) {
        try {
            StringBuilder res = new StringBuilder();
            int code = runScriptAsRoot(ctx, scriptHeader(ctx)
                    + "dmesg | $GREP DROIDWALL\n", res);
            if (code != 0) {
                if (res.length() == 0) {
                    res.append("Log is empty");
                }
                alert(ctx, res);
                return;
            }
            final BufferedReader r = new BufferedReader(new StringReader(
                    res.toString()));
            final Integer unknownUID = -99;
            res = new StringBuilder();
            String line;
            int start, end;
            Integer appid;
            final HashMap<Integer, LogInfo> map = new HashMap<Integer, LogInfo>();
            LogInfo loginfo = null;
            while ((line = r.readLine()) != null) {
                if (line.indexOf("[DROIDWALL]") == -1)
```



```
                continue;
            appid = unknownUID;
            if (((start = line.indexOf("UID=")) != -1)
                    && ((end = line.indexOf(" ", start)) != -1)) {
                appid = Integer.parseInt(line.substring(start + 4, end));
            }
            loginfo = map.get(appid);
            if (loginfo == null) {
                loginfo = new LogInfo();
                map.put(appid, loginfo);
            }
            loginfo.totalBlocked += 1;
            if (((start = line.indexOf("DST=")) != -1)
                    && ((end = line.indexOf(" ", start)) != -1)) {
                String dst = line.substring(start + 4, end);
                if (loginfo.dstBlocked.containsKey(dst)) {
                    loginfo.dstBlocked.put(dst,
                            loginfo.dstBlocked.get(dst) + 1);
                } else {
                    loginfo.dstBlocked.put(dst, 1);
                }
            }
        }
        final DroidApp[] apps = getApps(ctx);
        for (Integer id : map.keySet()) {
            res.append("App ID ");
            if (id != unknownUID) {
                res.append(id);
                for (DroidApp app : apps) {
                    if (app.uid == id) {
                        res.append(" (").append(app.names[0]);
                        if (app.names.length > 1) {
                            res.append(", ...)");
                        } else {
                            res.append(")");
                        }
                        break;
                    }
                }
            } else {
                res.append("(kernel)");
            }
            loginfo = map.get(id);
            res.append(" - Blocked ").append(loginfo.totalBlocked)
                    .append(" packets");
            if (loginfo.dstBlocked.size() > 0) {
                res.append(" (");
                boolean first = true;
                for (String dst : loginfo.dstBlocked.keySet()) {
                    if (!first) {
                        res.append(", ");
```

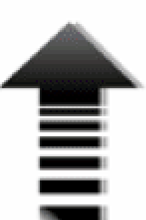
```
                        }
                        res.append(loginfo.dstBlocked.get(dst))
                                .append(" packets for ").append(dst);
                        first = false;
                    }
                    res.append(")");
                }
                res.append("\n\n");
            }
            if (res.length() == 0) {
                res.append("Log is empty");
            }
            alert(ctx, res);
        } catch (Exception e) {
            alert(ctx, "error: " + e);
        }
    }
```

(9) 编写函数 hasRootAccess()检查是否具备进入根目录的权限，具体代码如下：

```
    public static boolean hasRootAccess(Context ctx, boolean showErrors) {
        if (hasroot)
            return true;
        final StringBuilder res = new StringBuilder();
        try {
            // Run an empty script just to check root access
            if (runScriptAsRoot(ctx, "exit 0", res) == 0) {
                hasroot = true;
                return true;
            }
        } catch (Exception e) {
        }
        if (showErrors) {
            alert(ctx,
                    "Could not acquire root access.\n"
                            + "You need a rooted phone to run DroidWall.\n\n"
                            + "If this phone is already rooted, please
                                make sure DroidWall has enough
                                permissions to execute the \"su\"
                                command.\n"
                            + "Error message: " + res.toString());
        }
        return false;
    }
```

(10) 编写函数 runScript()执行前面编写的 Script 脚本头程序，此函数比较具有代表意义，能够在 Android 中调用并执行 Script 程序。函数 runScript()的具体代码如下：

```
    public static int runScript(Context ctx, String script, StringBuilder res,
            long timeout, boolean asroot) {
        final File file = new File(ctx.getDir("bin", 0), SCRIPT FILE);
       final ScriptRunner runner = new ScriptRunner(file, script, res, asroot);
```



```
        runner.start();
        try {
            if (timeout > 0) {
                runner.join(timeout);
            } else {
                runner.join();
            }
            if (runner.isAlive()) {
                // 设置超时
                runner.interrupt();
                runner.join(130);
                runner.destroy();
                runner.join(50);
            }
        } catch (InterruptedException ex) {
        }
        return runner.exitcode;
    }
```

(11) 编写函数 runScriptAsRoot()，功能是在 Root 权限下执行脚本程序。具体代码如下：

```
    public static int runScriptAsRoot(Context ctx, String script,
            StringBuilder res, long timeout) {
        return runScript(ctx, script, res, timeout, true);
    }
```

(12) 编写函数 runScript()，功能是设置普通用户权限执行脚本程序。具体代码如下：

```
    public static int runScript(Context ctx, String script, StringBuilder res)
            throws IOException {
        return runScript(ctx, script, res, 40000, false);
    }
```

(13) 编写函数 assertBinaries()，功能是判断二进制文件在高速缓存目录被安装。具体代码如下：

```
    public static boolean assertBinaries(Context ctx, boolean showErrors) {
        boolean changed = false;
        try {
            // 检查 iptables armv5 过滤包
            File file = new File(ctx.getDir("bin", 0), "iptables armv5");
            if (!file.exists()) {
                copyRawFile(ctx, R.raw.iptables armv5, file, "755");
                changed = true;
            }
            //检查 busybox
            file = new File(ctx.getDir("bin", 0), "busybox g1");
            if (!file.exists()) {
                copyRawFile(ctx, R.raw.busybox g1, file, "755");
                changed = true;
            }
            if (changed) {
                Toast.makeText(ctx, R.string.toast_bin_installed,
```

```
                        Toast.LENGTH LONG).show();
            }
        } catch (Exception e) {
            if (showErrors)
                alert(ctx, "Error installing binary files: " + e);
            return false;
        }
        return true;
    }
```

14.5.4 实现广播模块

编写文件 BootBroadcast.java，此文件是一个广播文件，在系统执行后将广播 ptables 规则。因为在规则中并没有设置开启显示信息，所以使用广播功能显示设置信息。文件 BootBroadcast.java 的主要代码如下：

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.widget.Toast;
public class BootBroadcast extends BroadcastReceiver {

    public void onReceive(final Context context, final Intent intent) {
        if (Intent.ACTION BOOT COMPLETED.equals(intent.getAction())) {
            if (Api.isEnabled(context)) {
                final Handler toaster = new Handler() {
                    public void handleMessage(Message msg) {
                        if (msg.arg1 != 0)
                            Toast.makeText(context, msg.arg1,
                                    Toast.LENGTH SHORT).show();
                    }
                };
                // 开启新线程阻止防火墙
                new Thread() {
                    @Override
                    public void run() {
                        if (!Api.applySavedIptablesRules(context, false)) {
                            // Error enabling firewall on boot
                            final Message msg = new Message();
                            msg.arg1 = R.string.toast error enabling;
                            toaster.sendMessage(msg);
                            Api.setEnabled(context, false);
                        }
                    }
                }.start();
            }
        }
    }
}
```

14.5.5　删除针对软件的设置规则

编写文件 PackageBroadcast.java，此文件也是一个具备广播功能的文件。当在手机中卸载一个软件后，会在防火墙中删除针对此软件的设置规则。文件 PackageBroadcast.java 的主要代码如下：

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class PackageBroadcast extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if (Intent.ACTION_PACKAGE_REMOVED.equals(intent.getAction())) {
            //忽略应用更新
            final boolean replacing = intent.getBooleanExtra
                (Intent.EXTRA REPLACING, false);
            if (!replacing) {
                final int uid = intent.getIntExtra(Intent.EXTRA UID, -123);
                Api.applicationRemoved(context, uid);
            }
        }
    }
}
```

14.5.6　登录验证

编写文件 PassDialog.java，功能是在输入密码对话框中获取用户输入的密码，只有输入合法的密码才能登录系统。文件 PassDialog.java 的主要代码如下：

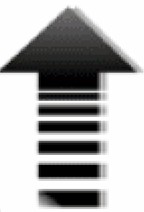
```
public class PassDialog extends Dialog implements
android.view.View.OnClickListener, android.view.View.OnKeyListener,
OnCancelListener {
    private final Callback callback;
    private final EditText pass;
    /**创建一个对话框*/
    public PassDialog(Context context, boolean setting, Callback callback) {
        super(context);
        final View view = getLayoutInflater().inflate(R.layout.pass_dialog, null);
        ((TextView)view.findViewById(R.id.pass message)).setText
            (setting ? R.string.enternewpass : R.string.enterpass);
        ((Button)view.findViewById(R.id.pass ok)).setOnClickListener(this);
        ((Button)view.findViewById(R.id.pass cancel)).setOnClickListener(this);
        this.callback = callback;
        this.pass = (EditText) view.findViewById(R.id.pass input);
        this.pass.setOnKeyListener(this);
        setTitle(setting ? R.string.pass_titleset : R.string.pass_titleget);
```

```
        setOnCancelListener(this);
        setContentView(view);
    }
    @Override
    public void onClick(View v) {
        final Message msg = new Message();
        if (v.getId() == R.id.pass ok) {
            msg.obj = this.pass.getText().toString();
        }
        dismiss();
        this.callback.handleMessage(msg);
    }
    @Override
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE ENTER) {
            final Message msg = new Message();
            msg.obj = this.pass.getText().toString();
            this.callback.handleMessage(msg);
            dismiss();
            return true;
        }
        return false;
    }
    @Override
    public void onCancel(DialogInterface dialog) {
        this.callback.handleMessage(new Message());
    }
}
```

14.5.7 打开或关闭某一个实施控件

编写文件 StatusWidget.java，功能是打开或关闭某一个实施控件。主要代码如下：

```
public class StatusWidget extends AppWidgetProvider {
    @Override
    public void onReceive(final Context context, final Intent intent) {
        super.onReceive(context, intent);
        if (Api.STATUS CHANGED MSG.equals(intent.getAction())) {
            // 当防火墙状态改变时马上广播信息
            final Bundle extras = intent.getExtras();
            if (extras != null && extras.containsKey(Api.STATUS EXTRA)) {
                final boolean firewallEnabled = extras
                        .getBoolean(Api.STATUS EXTRA);
                final AppWidgetManager manager = AppWidgetManager
                        .getInstance(context);
                final int[] widgetIds = manager
                        .getAppWidgetIds(new ComponentName(context,
                                StatusWidget.class));
                showWidget(context, manager, widgetIds, firewallEnabled);
            }
```



```
        } else if (Api.TOGGLE_REQUEST_MSG.equals(intent.getAction())) {
            // 根据防火墙开关信息广播状态信息
            final SharedPreferences prefs = context.getSharedPreferences(
                    Api.PREFS_NAME, 0);
           final boolean enabled = !prefs.getBoolean(Api.PREF_ENABLED, true);
            final String pwd = prefs.getString(Api.PREF_PASSWORD, "");
            if (!enabled && pwd.length() != 0) {
                Toast.makeText(context,
                        "Cannot disable firewall - password defined!",
                        Toast.LENGTH_SHORT).show();
                return;
            }
            final Handler toaster = new Handler() {
                public void handleMessage(Message msg) {
                    if (msg.arg1 != 0)
                       Toast.makeText(context, msg.arg1, Toast.LENGTH_SHORT)
                            .show();
                }
            };
            //开启新线程改变防火墙
            new Thread() {
                @Override
                public void run() {
                    final Message msg = new Message();
                    if (enabled) {
                        if (Api.applySavedIptablesRules(context, false)) {
                            msg.arg1 = R.string.toast_enabled;
                            toaster.sendMessage(msg);
                        } else {
                            msg.arg1 = R.string.toast_error_enabling;
                            toaster.sendMessage(msg);
                            return;
                        }
                    } else {
                        if (Api.purgeIptables(context, false)) {
                            msg.arg1 = R.string.toast_disabled;
                            toaster.sendMessage(msg);
                        } else {
                            msg.arg1 = R.string.toast_error_disabling;
                            toaster.sendMessage(msg);
                            return;
                        }
                    }
                    Api.setEnabled(context, enabled);
                }
            }.start();
        }
    }

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
```

```
            int[] ints) {
        super.onUpdate(context, appWidgetManager, ints);
        final SharedPreferences prefs = context.getSharedPreferences(
                Api.PREFS_NAME, 0);
        boolean enabled = prefs.getBoolean(Api.PREF_ENABLED, true);
        showWidget(context, appWidgetManager, ints, enabled);
    }

    private void showWidget(Context context, AppWidgetManager manager,
            int[] widgetIds, boolean enabled) {
            final RemoteViews views = new RemoteViews(context.getPackageName(),
                R.layout.onoff_widget);
        final int iconId = enabled ? R.drawable.widget_on
                : R.drawable.widget_off;
        views.setImageViewResource(R.id.widgetCanvas, iconId);
        final Intent msg = new Intent(Api.TOGGLE_REQUEST_MSG);
       final PendingIntent intent = PendingIntent.getBroadcast(context, -1,
                msg, PendingIntent.FLAG_UPDATE_CURRENT);
        views.setOnClickPendingIntent(R.id.widgetCanvas, intent);
        manager.updateAppWidget(widgetIds, views);
    }
}
```

到此为止，整个网络流量监控系统介绍完毕。

14.6 系 统 测 试

执行后的主界面效果如图 14-5 所示，按下 Menu 键后会弹出设置选项界面，如图 14-6 所示。

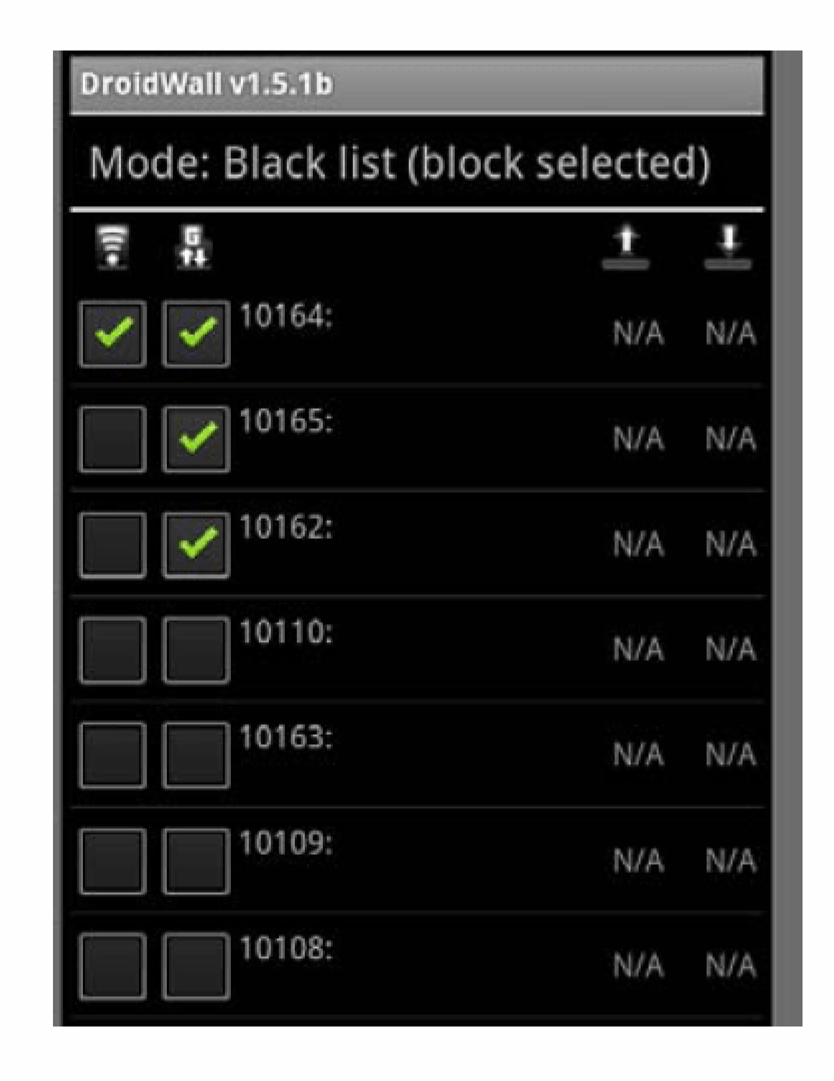


图 14-5 主界面



图 14-6 设置界面



单击设置界面中的 Firewall disabled 选项可以打开/关闭防火墙，单击设置界面中的 Log enabled 选项可以打开/关闭日志，单击设置界面中的 Save rules 选项会弹出保存进度条，如图 14-7 所示。

单击设置界面中的按钮会退出当前系统；单击设置界面中的按钮会弹出帮助界面，如图 14-8 所示；单击设置界面中的按钮会弹出一个新功能界面，如图 14-9 所示。在此界面中可以选择实现其他功能，例如单击 Set password 选项后会弹出一个设置密码界面，如图 14-10 所示。



图 14-7　保存规则进度条

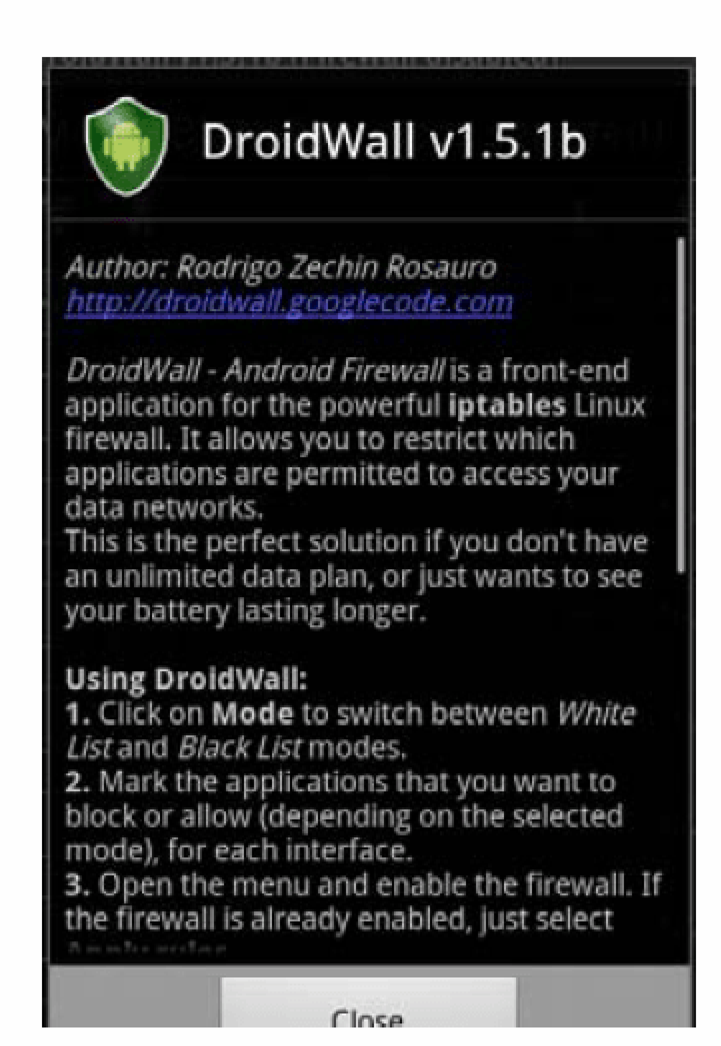


图 14-8　帮助界面



图 14-9　新功能界面

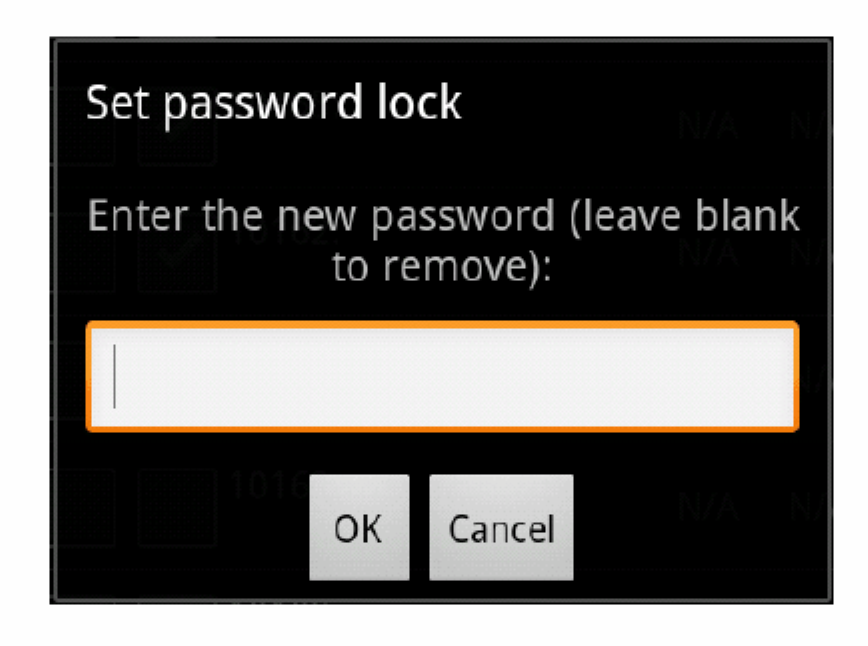


图 14-10　设置密码界面

Android

第 15 章

Android网络典型应用实践

经过本书前面内容的学习，Android 网络开发技术已经讲解完毕。本章将详细介绍 Android 在网络领域的常用模块，不但探讨了新技术，而且对前面的内容进行一个整体回顾，对所学知识进行一个拔高处理。希望通过对这些知识点的学习，使自己的水平得到一个质的提高。

15.1 测试网络下载速度

目前用户比较关心网速的问题，大家都不喜欢用极慢的速度下载自己心仪的电影。所以在 Android 平台中，很有必要开发一个网速测试程序。在接下来的演示代码中，将通过下载文件的大小和当前读取的字节数，在固定的时间中检测网络速度。

(1) 定义网络信息类 NetWorkSpeedInfo，其中设置了需要的常量值。

```
package cc.androidos.speed;
public class NetWorkSpeedInfo
{

   /**Network speed*/
   public  long speed = 0;
   /**Had finished bytes*/
   public  long hadFinishedBytes = 0;
   /**Total bytes of a file, default is 1024 bytes,1K*/
   public  long totalBytes = 1024;

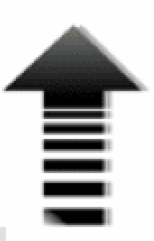
   /**The net work type, 3G or GSM and so on*/
   public  int networkType = 0;

   /**Down load the file percent 0----100*/
   public  int downloadPercent = 0;
}
```

(2) 编写一个名为 SpeedActivity 的主 Activity，实现获取速度操作。

```
package cc.androidos.speed;
import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
public class SpeedActivity extends Activity
{
   /** Called when the activity is first created. */

   TextView fileLength = null;
   TextView speed = null;
   TextView hasDown = null;
   TextView percent = null;
```



```
    String url = "";

    ImageView imageView = null;
    byte[] imageData = null;

    NetWorkSpeedInfo netWorkSpeedInfo = null;
    private final int UPDATE SPEED = 1;
    @Override
    public void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.main );

        hasDown = ( TextView ) findViewById( R.id.hasDown );
        fileLength = ( TextView ) findViewById( R.id.fileLength );
        speed = ( TextView ) findViewById( R.id.speed );
        percent = ( TextView ) findViewById( R.id.percent );
        imageView = ( ImageView ) findViewById( R.id.ImageView01 );
        Button b = ( Button ) findViewById( R.id.Button01 );
        url = getString( R.string.image url );
        netWorkSpeedInfo = new NetWorkSpeedInfo();
        b.setOnClickListener( new View.OnClickListener()
        {
            @Override
            public void onClick( View arg0 )
            {

                //down load the file thread
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        imageData = ReadFile.getFileFromUrl( url,
                            netWorkSpeedInfo );
                        stop();
                    }
                }.start();

                //get the speed , down load bytes ,update the view thread
                new Thread()
                {
                    @Override
                    public void run()
                    {

                        while ( netWorkSpeedInfo.hadFinishedBytes <
                            netWorkSpeedInfo.totalBytes )
                        {
                            netWorkSpeedInfo.downloadPercent = ( int )
                               (( ( double ) netWorkSpeedInfo.hadFinishedBytes /
```




```
                    ( double ) netWorkSpeedInfo.totalBytes ) * 100);
                try
                {
                    sleep( 1500 );
                }
                catch ( InterruptedException e )
                {
                    e.printStackTrace();
                }

                Log.e( "update,send the message to update", "" );
                //update view
                handler.sendEmptyMessage( UPDATE SPEED );
            }

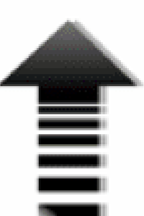
            //finished
            if( netWorkSpeedInfo.hadFinishedBytes ==
                 netWorkSpeedInfo.totalBytes )
            {

                netWorkSpeedInfo.downloadPercent = ( int )
                   (( ( double ) netWorkSpeedInfo.hadFinishedBytes /
                    ( double ) netWorkSpeedInfo.totalBytes ) * 100);
                handler.sendEmptyMessage( UPDATE SPEED );
                Log.e( "update",
                    ",send the message to update and stop" );
                stop();
            }

        }
    }.start();
}
} );
}



/**
 * Handler for post message into OS
 */
private Handler handler = new Handler()
{
    @Override
    public void handleMessage( Message msg )
    {
        int value = msg.what;
        switch ( value )
        {
            case UPDATE SPEED:
                updateView();
                break;
```



```
                default:
                    break;
            }
        }
    };

    /**
     * Update the view method
     */
    private void updateView()
    {
        speed.setText( netWorkSpeedInfo.speed + "bytes/s" );
        hasDown.setText( netWorkSpeedInfo.hadFinishedBytes + "bytes" );
        fileLength.setText( netWorkSpeedInfo.totalBytes + "" );
        percent.setText( netWorkSpeedInfo.downloadPercent+"%" );
        if( imageData != null )
        {
            Bitmap b = BitmapFactory.decodeByteArray( imageData, 0,
                imageData.length );
            imageView.setImageBitmap( b );
        }
    }
}
```

(3) 编写读取指定 Web 文件的代码，用于通过读取指定的文件来测试速度。

```
package cc.androidos.speed;
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;
import android.util.Log;
public class ReadFil{
    public static byte[] getFileFromUrl( String url,NetWorkSpeedInfo
        netWorkSpeedInfo )
    {
        int currentByte = 0;
        int fileLength = 0;
        long startTime = 0;
        long intervalTime = 0;
        byte[] b = null;

        int bytecount = 0;
        URL urlx = null;
        URLConnection con = null;
        InputStream stream = null;
        tr        {
            Log.d( "URL:", url );
            urlx = new URL( url );
            con = urlx.openConnection();
            con.setConnectTimeout( 20000 );
            con.setReadTimeout( 20000 );
```

```
            fileLength = con.getContentLength();
            stream = con.getInputStream();
            netWorkSpeedInfo.totalBytes = fileLength;
            b = new byte[fileLength];
            startTime = System.currentTimeMillis();
            while ( ( currentByte = stream.read() ) != -1 )
            {
                netWorkSpeedInfo.hadFinishedBytes++;
                intervalTime = System.currentTimeMillis() - startTime;
                if(intervalTime==0){
                    netWorkSpeedInfo.speed = 1000;
                }else{
                    netWorkSpeedInfo.speed = (  netWorkSpeedInfo.
                        hadFinishedBytes / intervalTime ) * 1000;
                }
                if(bytecount<fileLength){
                    b[bytecount++] = ( byte ) currentByte;
                }
            }
        }
        catch ( Exception e )
        {
            Log.e( "exception : ", e.getMessage()+"" );
        }
        finall       {
            tr          {
                if( stream != null )
                {
                    stream.close();
                }
            }
            catch ( Exception e )
            {
                Log.e( "exception : ", e.getMessage() );
            }
        }
        return b;
    }
}
```

15.2 通过 Handler 实现异步消息处理

在 Android 系统中，可以与服务端实现 HTTP 通信，并解析 XML 数据，通过 Handler 实现异步消息处理。在接下来的演示代码中，分别实现了以下三个重要操作。

(1) HTTP 通信：与服务端做 HTTP 通信，分别以 Get 方式和 Post 方式做演示。

(2) XML 解析：可以用两种方式解析 XML，分别是 DOM 方式和 SAX 方式。



(3) 异步消息处理：通过 Handler 实现异步消息处理，以一个自定义的异步下载类来说明 Handler 的用法。

15.2.1　实现HTTP通信和XML解析的演示

(1) 定义一个名为 MySAXHandler 的类，此类继承于 DefaultHandler，功能是实现指定 XML 的 SAX 解析器。并且在下面的代码中使用了 SAX 流式解析方式，通过事件模型解析 XML，这种方式只能顺序解析。

```
package com.webabcd.communication;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
// 采用 DOM - W3C 标准，需要把 XML 数据全部加载完成后才能对其做解析，可对树做任意遍历
public class MySAXHandler extends DefaultHandler {

    private boolean mIsTitleTag = false;
    private boolean mIsSalaryTag = false;
    private boolean mIsBirthTag = false;
    private String mResult = "";

    // 打开 XML 文档的回调函数
    @Override
    public void startDocument() throws SAXException {
        // TODO Auto-generated method stub
        super.startDocument();
    }

    // 关闭 XML 文档的回调函数
    @Override
    public void endDocument() throws SAXException {
        // TODO Auto-generated method stub
        super.endDocument();
    }

    // 发现元素开始标记就回调此函数
    @Override
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {
        if (localName == "title")
            mIsTitleTag = true;
        else if (localName == "salary")
            mIsSalaryTag = true;
        else if (localName == "dateOfBirth")
            mIsBirthTag = true;
        else if (localName == "employee")
            mResult += "\nname:" + attributes.getValue("name");
    }
```

```
    // 发现元素结束标记就回调此函数
    @Override
    public void endElement(String uri, String localName, String qName)
            throws SAXException {
        if (localName == "title")
            mIsTitleTag = false;
        else if (localName == "salary")
            mIsSalaryTag = false;
        else if (localName == "dateOfBirth")
            mIsBirthTag = false;
    }

    // 发现元素值或属性值就回调此函数
    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {
        if (mIsTitleTag)
            mResult += new String(ch, start, length);
        else if (mIsSalaryTag)
            mResult += " salary:" + new String(ch, start, length);
        else if (mIsBirthTag)
            mResult += " dateOfBirth:" + new String(ch, start, length);
    }

    public String getResult(){
        return mResult;
    }
}
```

(2) 在下面的演示代码中主要定义了以下两个核心方法。

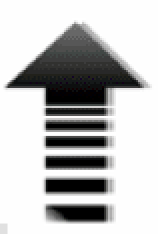
方法 httpGetDemo()：以 HTTP 协议的 get()方法获取远程页面响应的内容。

方法 httpPostDemo()：以 HTTP 协议的 post()方法向远程页面传递参数，并获取其响应的内容。

```
package com.webabcd.communication;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
```



```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;
import org.apache.http.util.ByteArrayBuffer;
import org.apache.http.util.EncodingUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class Main extends Activity {

   private TextView textView;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      textView = (TextView) this.findViewById(R.id.textView);

      Button btn1 = (Button) this.findViewById(R.id.btn1);
      btn1.setText("http get demo");
      btn1.setOnClickListener(new Button.OnClickListener() {
         public void onClick(View v) {
            httpGetDemo();
         }
      });

      Button btn2 = (Button) this.findViewById(R.id.btn2);
      btn2.setText("http post demo");
      btn2.setOnClickListener(new Button.OnClickListener() {
         public void onClick(View v) {
            httpPostDemo();
         }
      });
```




```
    Button btn3 = (Button) this.findViewById(R.id.btn3);
    // DOM - Document Object Model
    btn3.setText("DOM 解析 XML");
    btn3.setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v) {
            DOMDemo();
        }
    });

    Button btn4 = (Button) this.findViewById(R.id.btn4);
    // SAX - Simple API for XML
    btn4.setText("SAX 解析 XML");
    btn4.setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v) {
            SAXDemo();
        }
    });
}

// Android 调用 http 协议的 Get 方法
// 本例：以 HTTP 协议的 Get 方法获取远程页面响应的内容
private void httpGetDemo(){
    try {
        // 模拟器测试时，请使用外网地址
        URL url = new URL("http://xxx.xxx.xxx");
        URLConnection con = url.openConnection();

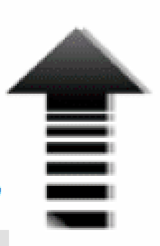
        String result = "http status code: " +
            ((HttpURLConnection)con).getResponseCode() + "\n";
        // HttpURLConnection.HTTP OK

        InputStream is = con.getInputStream();
        BufferedInputStream bis = new BufferedInputStream(is);
        ByteArrayBuffer bab = new ByteArrayBuffer(32);
        int current = 0;
        while ( (current = bis.read()) != -1 ){
            bab.append((byte)current);
        }
        result += EncodingUtils.getString(bab.toByteArray(), HTTP.UTF_8);

        bis.close();
        is.close();

        textView.setText(result);
    } catch (Exception e) {
        textView.setText(e.toString());
    }
}

// Android 调用 HTTP 协议的 Post 方法
```



```
// 本例：以 HTTP 协议的 Post 方法向远程页面传递参数，并获取其响应的内容
private void httpPostDemo(){
    try {
        // 模拟器测试时，请使用外网地址
        String url = "http://5billion.com.cn/post.php";
        Map<String, String> data = new HashMap<String, String>();
        data.put("name", "webabcd");
        data.put("salary", "100");

        DefaultHttpClient httpClient = new DefaultHttpClient();
        HttpPost httpPost = new HttpPost(url);
        ArrayList<BasicNameValuePair> postData = new
            ArrayList<BasicNameValuePair>();
        for (Map.Entry<String, String> m : data.entrySet()) {
           postData.add(new BasicNameValuePair(m.getKey(),
               m.getValue()));
        }

        UrlEncodedFormEntity entity = new
            UrlEncodedFormEntity(postData, HTTP.UTF 8);
        httpPost.setEntity(entity);

        HttpResponse response = httpClient.execute(httpPost);

        String result = "http status code: " +
            response.getStatusLine().getStatusCode() + "\n";
        // HttpURLConnection.HTTP OK

        HttpEntity httpEntity = response.getEntity();

        InputStream is = httpEntity.getContent();
        result += convertStreamToString(is);

        textView.setText(result);
    } catch (Exception e) {
        textView.setText(e.toString());
    }
}

// 以 DOM 方式解析 XML
private void DOMDemo(){
    try   {
        DocumentBuilderFactory docFactory =
            DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
        Document doc = docBuilder.parse(this.getResources().
            openRawResource(R.raw.employee));
        Element rootElement = doc.getDocumentElement();
        NodeList employeeNodeList = rootElement.getElementsByTagName
           ("employee");
```

```
            textView.setText("DOMDemo" + "\n");
            String title = rootElement.getElementsByTagName ("title").
                item(0).getFirstChild().getNodeValue();
            textView.append(title);
            for (int i=0; i<employeeNodeList.getLength(); i++){
                Element employeeElement = ((Element)employeeNodeList.item(i));
                String name = employeeElement.getAttribute("name");
                String salary = employeeElement.getElementsByTagName
                    ("salary").item(0).getFirstChild().getNodeValue();
                String dateOfBirth = employeeElement.getElementsByTagName
                    ("dateOfBirth").item(0).getFirstChild().getNodeValue();
                textView.append("\nname: "+name+" salary: "+salary+"
                    dateOfBirth: " + dateOfBirth);
            }
        } catch (Exception e) {
            textView.setText(e.toString());
        }
    }

    // 以 SAX 方式解析 XML
    private void SAXDemo(){
        try   {
            SAXParserFactory saxFactory = SAXParserFactory.newInstance();
            SAXParser parser = saxFactory.newSAXParser();
            XMLReader reader = parser.getXMLReader();

            MySAXHandler handler = new MySAXHandler();
            reader.setContentHandler(handler);
            reader.parse(new InputSource(this.getResources().openRawResource
                (R.raw.employee)));
            String result = handler.getResult();
            textView.setText("SAXDemo" + "\n");
            textView.append(result);
        } catch (Exception e) {
            textView.setText(e.toString());
        }
    }

    // 辅助方法，用于把流转换为字符串
    private String convertStreamToString(InputStream is) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(is));
        StringBuilder sb = new StringBuilder();

        String line = null;
        try {
            while ((line = reader.readLine()) != null) {
                sb.append(line + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
```

```
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            return sb.toString();
        }
}
```

15.2.2　使用Handler实现异步消息处理

当用 Handler 实现异步消息处理时，在接下来的演示代码中，以一个可以实时汇报下载进度的异步下载类为例，开发了一个 Android 类库。在以下演示代码中此类库的名字为 webabcd_util。打开 Eclipse，依次选择 New | Java Project 命令，然后在项目上右击并依次选择 Build Path | Add Libraries | User Library | User Libraries | New 命令，为类库起一个名字，然后选中这个类库，单击 Add JARs 导入 Android 的 jar 包。最后在项目上右击，依次选择 Build Path | Add Libraries | User Library 命令，选择 Android 库。

```
package webabcd.util;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;

import org.apache.http.protocol.HTTP;

import android.os.Handler;
import android.os.Message;
import android.util.Log;

// 以一个异步下载实例，来演示 Android 的异步消息处理(用 Handler 的方式)
public class DownloadManagerAsync {

    public DownloadManagerAsync() {

    }

    // 实例化自定义的 Handler
    EventHandler mHandler = new EventHandler(this);

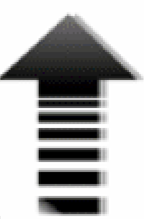
    // 按指定 URL 地址下载文件到指定路径
    public void download(final String url, final String savePath) {
        new Thread(new Runnable() {
            public void run() {
```

```
                try {
                    sendMessage(FILE DOWNLOAD CONNECT);
                    URL sourceUrl = new URL(url);
                    URLConnection conn = sourceUrl.openConnection();
                    InputStream inputStream = conn.getInputStream();
                    int fileSize = conn.getContentLength();
                    File savefile = new File(savePath);
                    if (savefile.exists()) {
                        savefile.delete();
                    }
                    savefile.createNewFile();

                    FileOutputStream outputStream = new FileOutputStream(
                            savePath, true);
                    byte[] buffer = new byte[1024];
                    int readCount = 0;
                    int readNum = 0;
                    int prevPercent = 0;
                    while (readCount < fileSize && readNum != -1) {
                        readNum = inputStream.read(buffer);
                        if (readNum > -1) {
                            outputStream.write(buffer);
                            readCount = readCount + readNum;

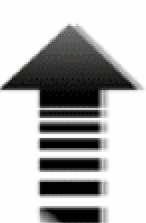
                            int percent = (int) (readCount * 100 / fileSize);
                            if (percent > prevPercent) {
                                // 发送下载进度信息
                                sendMessage(FILE DOWNLOAD UPDATE, percent,
                                        readCount);

                                prevPercent = percent;
                            }
                        }
                    }
                    outputStream.close();
                    sendMessage(FILE DOWNLOAD COMPLETE, savePath);
                } catch (Exception e) {
                    sendMessage(FILE DOWNLOAD ERROR, e);
                    Log.e("MyError", e.toString());
                }
            }
        }).start();
    }
    // 读取指定 URL 地址的响应内容
    public void download(final String url) {
        new Thread(new Runnable() {
            public void run() {
                try {
                    sendMessage(FILE DOWNLOAD CONNECT);
                    URL sourceUrl = new URL(url);
                    URLConnection conn = sourceUrl.openConnection();
```



```
                conn.setConnectTimeout(3000);
                BufferedReader reader = new BufferedReader(
                        new InputStreamReader(conn.getInputStream(),
                                HTTP.UTF 8));
                String line = null;
                StringBuffer content = new StringBuffer();
                while ((line = reader.readLine()) != null) {
                    content.append(line);
                }
                reader.close();
                sendMessage(FILE DOWNLOAD COMPLETE, content.toString());
            } catch (Exception e) {
                sendMessage(FILE DOWNLOAD ERROR, e);
                Log.e("MyError", e.toString());
            }
        }
    }).start();
}
// 向 Handler 发送消息
private void sendMessage(int what, Object obj) {
    // 构造需要向 Handler 发送的消息
    Message msg = mHandler.obtainMessage(what, obj);
    // 发送消息
    mHandler.sendMessage(msg);
}
private void sendMessage(int what) {
    Message msg = mHandler.obtainMessage(what);
    mHandler.sendMessage(msg);
}
private void sendMessage(int what, int arg1, int arg2) {
    Message msg = mHandler.obtainMessage(what, arg1, arg2);
    mHandler.sendMessage(msg);
}
private static final int FILE DOWNLOAD CONNECT = 0;
private static final int FILE DOWNLOAD UPDATE = 1;
private static final int FILE DOWNLOAD COMPLETE = 2;
private static final int FILE DOWNLOAD ERROR = -1;
// 自定义的 Handler
private class EventHandler extends Handler {
    private DownloadManagerAsync mManager;
    public EventHandler(DownloadManagerAsync manager) {
        mManager = manager;
    }
    // 处理接收到的消息
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case FILE DOWNLOAD CONNECT:
            if (mOnDownloadConnectListener != null)
                mOnDownloadConnectListener.onDownloadConnect(mManager);
            break;
```

```
            case FILE DOWNLOAD UPDATE:
                if (mOnDownloadUpdateListener != null)
                    mOnDownloadUpdateListener.onDownloadUpdate(mManager,
                            msg.arg1);
                break;
            case FILE DOWNLOAD COMPLETE:
                if (mOnDownloadCompleteListener != null)
                    mOnDownloadCompleteListener.onDownloadComplete(mManager,
                            msg.obj);
                break;
            case FILE DOWNLOAD ERROR:
                if (mOnDownloadErrorListener != null)
                    mOnDownloadErrorListener.onDownloadError(mManager,
                            (Exception) msg.obj);
                break;
            default:
                break;
            }
        }
    }
    // 定义连接事件
    private OnDownloadConnectListener mOnDownloadConnectListener;
    public interface OnDownloadConnectListener {
        void onDownloadConnect(DownloadManagerAsync manager);
    }
    public void setOnDownloadConnectListener(OnDownloadConnectListener listener) {
        mOnDownloadConnectListener = listener;
    }
    // 定义下载进度更新事件
    private OnDownloadUpdateListener mOnDownloadUpdateListener;
    public interface OnDownloadUpdateListener {
        void onDownloadUpdate(DownloadManagerAsync manager, int percent);
    }
    public void setOnDownloadUpdateListener(OnDownloadUpdateListener listener) {
        mOnDownloadUpdateListener = listener;
    }
    // 定义下载完成事件
    private OnDownloadCompleteListener mOnDownloadCompleteListener;
    public interface OnDownloadCompleteListener {
        void onDownloadComplete(DownloadManagerAsync manager, Object result);
    }
    public void setOnDownloadCompleteListener(
            OnDownloadCompleteListener listener) {
        mOnDownloadCompleteListener = listener;
    }
    // 定义下载异常事件
    private OnDownloadErrorListener mOnDownloadErrorListener;
    public interface OnDownloadErrorListener {
        void onDownloadError(DownloadManagerAsync manager, Exception e);
    }
    public void setOnDownloadErrorListener(OnDownloadErrorListener
```



```
listener) {
        mOnDownloadErrorListener = listener;
    }
}
```

然后调用上面的自定义的 Android 类库，在项目上右击，依次选择 Properties | Java Build Path | Projects | Add 命令来引用上面的类库。

```
package com.webabcd.handler;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

import webabcd.util.DownloadManagerAsync;

public class Main extends Activity implements
        DownloadManagerAsync.OnDownloadCompleteListener,
        DownloadManagerAsync.OnDownloadUpdateListener,
        DownloadManagerAsync.OnDownloadErrorListener {

    TextView txt;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DownloadManagerAsync manager = new DownloadManagerAsync();
        manager.setOnDownloadCompleteListener(this);
        manager.setOnDownloadUpdateListener(this);
        manager.download("http://files.cnblogs.com/webabcd/Android.rar",
            "/sdcard/Android.rar");

        txt = (TextView) this.findViewById(R.id.txt);
        txt.setText("开始下载");
    }

    public void onDownloadComplete(DownloadManagerAsync manager, Object result) {

        txt.setText("下载完成");
    }

    public void onDownloadUpdate(DownloadManagerAsync manager, int percent) {

        txt.setText("下载进度：" + String.valueOf(percent) + "%");
    }

    public void onDownloadError(DownloadManagerAsync manager, Exception e) {
        txt.setText("下载出错");
```

```
    }
}
```

15.3 实现网络多线程断点下载

在现实应用中，直接使用单线程下载 HTTP 文件对我们来说是一件非常简单的事。其实我们可以尝试使用多线程断点进行下载。本节将通过一个具体实例来讲解在 Android 中实现网络多线程断点下载的方法。

15.3.1 实现原理

要想从文件的指定位置处开始下载文件，可以通过 HTTP 请求信息头来设置，需要设置 HTTP 请求信息头的 Range 属性。在解决了多线程下载问题后，接下来开始解决支持断点下载的问题。其实很简单，只需将下载的进度保存到文件中即可，但是在 Android 中却不能这么做。在 Android 平台中，我们需要向文件中写入下载的文件数据，还需要向另一个文件中写入下载进度，这样会导致有一个文件的内容没有被写入的错误。所以我们就不能以文件的方式来保存下载进度，但可以通过数据库的方式保存下载进度。

15.3.2 具体实现

(1) 创建 Android 工程。

Project name：MulThreadDownloader。

BuildTarget：Android 4.0。

Application name：多线程断点下载。

Package name：com.changcheng.download。

Create Activity：MulThreadDownloader。

Min SDK Version：11。

(2) 编写文件 AndroidManifest.xml，在此文件中主要设置以下三个权限。

- 在 SDCard 中创建与删除文件权限。
- 往 SDCard 中写入数据权限。
- 访问 Internet 权限。

具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.changcheng.download"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon"
        android:label="@string/app_name">
      <activity android:name=".MulThreadDownloader"
```

```
                android:label="@string/app name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="20" />
        <!-- 在 SDCard 中创建与删除文件权限 -->
        <uses-permission android:name="android.permission.MOUNT
            UNMOUNT FILESYSTEMS"/>
        <!-- 往 SDCard 中写入数据权限 -->
        <uses-permission android:name="android.permission.WRITE
            EXTERNAL STORAGE"/>
        <!-- 访问 Internet 权限 -->
        <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

(3) 编写文件 strings.xml，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, DownloadActivity!</string>
    <string name="app name">多线程断点下载</string>
    <string name="path">下载路径</string>
    <string name="downloadbutton">下载</string>
    <string name="sdcarderror">SDCard 不存在或者写保护</string>
</resources>
```

(4) 编写 UI 布局文件 main.xml，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
    <!-- 下载路径 -->
        <TextView
            android:layout width="fill parent"
            android:layout height="wrap content"
            android:text="@string/path"
            />
    <EditText
            android:layout width="fill parent"
            android:layout height="wrap content"
            android:text="http://www.winrar.com.cn/download/wrar380sc.exe"
            android:id="@+id/path"
            />
        <!-- 下载按钮 -->
        <Button
```

```
            android:layout width="wrap content"
        android:layout height="wrap content"
        android:text="@string/downloadbutton"
        android:id="@+id/button"
        />
      <!-- 进度条 -->
      <ProgressBar
            android:layout width="fill parent"
        android:layout height="20dip"
        style="?android:attr/progressBarStyleHorizontal"
           android:id="@+id/downloadbar"/>
    <!-- 进度% -->
    <TextView
        android:layout width="fill parent"
        android:layout height="wrap content"
        android:gravity="center"
        android:id="@+id/resultView"
        />
</LinearLayout>
```

(5) 编写文件 MulThreadDownloader.java，具体代码如下：

```
package com.changcheng.download;
import java.io.File;
import com.changcheng.net.download.DownloadProgressListener;
import com.changcheng.net.download.FileDownloader;
import com.changcheng.download.R;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
public class MulThreadDownloader extends Activity {
       private EditText pathText;
       private ProgressBar progressBar;
       private TextView resultView;
       private Handler handler = new Handler(){
               @Override
               public void handleMessage(Message msg) {
                      if(!Thread.currentThread().isInterrupted()){
                             switch (msg.what) {
                             case 1:
                                // 获取当前文件下载的进度
                                int size = msg.getData().getInt("size");
                                progressBar.setProgress(size);
```

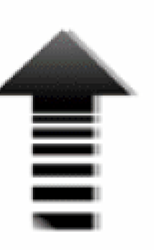
```
                        int result = (int)(((float)size/(float)
                            progressBar.getMax()) * 100);
                        resultView.setText(result+ "%");
                        if(progressBar.getMax() == size){
                            Toast.makeText(MulThreadDownloader.
                                this, "文件下载完成", 1).show();
                        }
                        break;
                    case -1:
                        String error = msg.getData().
                            getString("error");
                        Toast.makeText(MulThreadDownloader.
                            this, error, 1).show();
                        break;
                    }
                }
                super.handleMessage(msg);
            }
        };
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        pathText = (EditText)this.findViewById(R.id.path);
        progressBar = (ProgressBar)this.findViewById(R.id.downloadbar);
        resultView = (TextView)this.findViewById(R.id.resultView);
        Button button = (Button)this.findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            String path = pathText.getText().toString();
            if(Environment.getExternalStorageState().equals
              (Environment.MEDIA_MOUNTED)){
             //下载文件需要很长的时间，主线程是不能够长时间被阻塞，如果主线程被长时间
               阻塞，那么 Android 被回收应用
            download(path, Environment.getExternalStorageDirectory());
            }else{
                Toast.makeText(MulThreadDownloader.this,
                    R.string.sdcarderror, 1).show();
                }
            }
        });
    }
    /**
     * 下载文件
     * @param path 下载路径
     * @param saveDir 文件保存目录
     */
  //对于 Android 的 UI 控件，只能由主线程负责显示界面的更新，其他线程不能直接更新 UI 控件的显示
    public void download(final String path, final File saveDir){
       new Thread(new Runnable() {
```




```
                        @Override
                        public void run() {
                           FileDownloader downer = new FileDownloader
                              (MulThreadDownloader.this, path, saveDir, 3);
                        progressBar.setMax(downer.getFileSize());
                             //设置进度条的最大刻度
                        try {
                            downer.download(new DownloadProgressListener(){
                            @Override
                            public void onDownloadSize(int size) {
                            Message msg = new Message();
                            msg.what = 1;
                            msg.getData().putInt("size", size);
                            handler.sendMessage(msg);//发送消息
                            }});
                       } catch (Exception e) {
                            Message msg = new Message();
                            msg.what = -1;
                            msg.getData().putString("error", "下载失败");
                            handler.sendMessage(msg);
                    }
               }
         }).start();
    }
}
```

(6) 编写文件 FileDownload.java，具体代码如下：

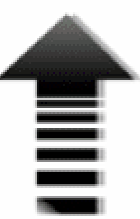
```
package com.changcheng.net.download;
import java.io.File;
import java.io.RandomAccessFile;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.ConcurrentHashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.changcheng.download.service.FileService;
import android.content.Context;
import android.util.Log;
/**
* 文件下载器
*/
public class FileDownloader {
        private Context context;
        private FileService fileService;
        private static final String TAG = "FileDownloader";
        /* 已下载文件大小 */
        private int downloadSize = 0;
```



```
/* 原始文件大小 */
private int fileSize = 0;
/* 线程数 */
private DownloadThread[] threads;
/* 下载路径  */
private URL url;
/* 本地保存文件 */
private File saveFile;
/* 下载记录文件 */
private File logFile;
/* 缓存各线程最后下载的位置*/
private Map<Integer, Integer> data = new
    ConcurrentHashMap<Integer, Integer>();
/* 每条线程下载的大小 */
private int block;
private String downloadUrl;//下载路径
/**
 * 获取线程数
 */
public int getThreadSize() {
        return threads.length;
}
/**
 * 获取文件大小
 * @return
 */
public int getFileSize() {
        return fileSize;
}
/**
 * 累计已下载大小
 * @param size
 */
protected synchronized void append(int size) {
        downloadSize += size;
}
/**
 * 更新指定线程最后下载的位置
 * @param threadId 线程 ID
 * @param pos 最后下载的位置
 */
protected void update(int threadId, int pos) {
    this.data.put(threadId, pos);
}
/**
 * 保存记录文件
 */
protected synchronized void saveLogFile() {
    this.fileService.update(this.downloadUrl, this.data);
}
/**
```

```
 * 构建文件下载器
 * @param downloadUrl 下载路径
 * @param fileSaveDir 文件保存目录
 * @param threadNum 下载线程数
 */
public FileDownloader(Context context, String downloadUrl, File
    fileSaveDir, int threadNum) {
    try {
        this.context = context;
        this.downloadUrl = downloadUrl;
        fileService = new FileService(context);
        this.url = new URL(downloadUrl);
        if(!fileSaveDir.exists()) fileSaveDir.mkdirs();
        this.threads = new DownloadThread[threadNum];
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setConnectTimeout(6*1000);
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept", "image/gif, image/jpeg,
            image/pjpeg, image/pjpeg, application/x-shockwave-flash,
            application/xaml+xml, application/vnd.ms-xpsdocument,
            application/x-ms-xbap, application/x-ms-application,
            application/vnd.ms-excel, application/vnd.ms-powerpoint,
            application/msword, */*");
        conn.setRequestProperty("Accept-Language", "zh-CN");
        conn.setRequestProperty("Referer", downloadUrl);
        conn.setRequestProperty("Charset", "UTF-8");
        conn.setRequestProperty("User-Agent", "Mozilla/4.0
             (compatible; MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET
             CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET
             CLR 3.0.4506.2152; .NET CLR 3.5.30729)");
        conn.setRequestProperty("Connection", "Keep-Alive");
        conn.connect();
        printResponseHeader(conn);
        if (conn.getResponseCode()==200) {
        this.fileSize = conn.getContentLength();//根据响应获取文件大小
        if (this.fileSize <= 0) throw new RuntimeException
            ("无法获知文件大小 ");

        String filename = getFileName(conn);
        this.saveFile = new File(fileSaveDir, filename);/* 保存文件 */
        Map<Integer, Integer> logdata = fileService.getData(downloadUrl);
        if(logdata.size()>0){
             data.putAll(logdata);
        }
        this.block = this.fileSize / this.threads.length + 1;
        if(this.data.size()==this.threads.length){
           for (int i = 0; i < this.threads.length; i++) {
                this.downloadSize += this.data.get(i+1)-(this.block * i);
        }
        print("已经下载的长度"+ this.downloadSize);
        }
```



```
            }else{
                throw new RuntimeException("服务器响应错误 ");
            }
        } catch (Exception e) {
            print(e.toString());
            throw new RuntimeException("连接不到下载路径 ");
        }
    }
    /**
     * 获取文件名
     */
    private String getFileName(HttpURLConnection conn) {
        String filename = this.url.toString().substring
            (this.url.toString().lastIndexOf('/') + 1);
        if(filename==null || "".equals(filename.trim())){
            //如果获取不到文件名称
            for (int i = 0;; i++) {
                String mine = conn.getHeaderField(i);
                if (mine == null) break;
                if("content-disposition".equals(conn.
                    getHeaderFieldKey(i).toLowerCase())){
                    Matcher m = Pattern.compile(".*filename=(.*)").
                        matcher(mine.toLowerCase());
                    if(m.find()) return m.group(1);
                    }
                }
                filename = UUID.randomUUID()+ ".tmp";//默认取一个文件名
            }
            return filename;
    }
    /**
     *  开始下载文件
     * @param listener 监听下载数量的变化，如果不需要了解实时下载的数量，可以
        设置为 null
     * @return 已下载文件大小
     * @throws Exception
     */
    public int download(DownloadProgressListener listener) throws
       Exception{
            try {
                if(this.data.size() != this.threads.length){
                    this.data.clear();
                for (int i = 0; i < this.threads.length; i++) {
                      this.data.put(i+1, this.block * i);
                }
            }
                for (int i = 0; i < this.threads.length; i++) {
                      int downLength = this.data.get(i+1) - (this.block * i);
                if(downLength < this.block && this.data.get(i+1)
                   <this.fileSize){ //该线程未完成下载时，继续下载
                   RandomAccessFile randOut = new RandomAccessFile
```




```
                    (this.saveFile, "rw");
                if(this.fileSize>0) randOut.setLength(this.fileSize);
                randOut.seek(this.data.get(i+1));
                this.threads[i] = new DownloadThread(this, this.url,
                    randOut, this.block, this.data.get(i+1), i+1);
                this.threads[i].setPriority(7);
                this.threads[i].start();
            }else{
                this.threads[i] = null;
            }
        }
                this.fileService.save(this.downloadUrl, this.data);
                boolean notFinish = true;//下载未完成
                while (notFinish) {// 循环判断是否下载完毕
                   Thread.sleep(900);
                   notFinish = false;//假定下载完成
                   for (int i = 0; i < this.threads.length; i++){
                   if (this.threads[i] != null && !this.threads[i].
                        isFinish()) {
                        notFinish = true;//下载没有完成
                   if(this.threads[i].getDownLength() == -1){
                       //如果下载失败，再重新下载
               RandomAccessFile randOut = new RandomAccessFile
                  (this.saveFile, "rw");
               randOut.seek(this.data.get(i+1));

               this.threads[i] = new DownloadThread(this, this.url,
                   randOut, this.block, this.data.get(i+1), i+1);
               this.threads[i].setPriority(7);
               this.threads[i].start();
               }
         }
    }
    if(listener!=null) listener.onDownloadSize(this.downloadSize);
    }
        fileService.delete(this.downloadUrl);
        } catch (Exception e) {
            print(e.toString());
            throw new Exception("下载失败");
        }
        return this.downloadSize;
}
/**
 * 获取 HTTP 响应头字段
 * @param http
 * @return
 */
public static Map<String, String> getHttpResponseHeader
     (HttpURLConnection http) {
Map<String, String> header = new LinkedHashMap<String, String>();
for (int i = 0;; i++) {
```

```
                String mine = http.getHeaderField(i);
                if (mine == null) break;
                header.put(http.getHeaderFieldKey(i), mine);
             }
             return header;
        }
        /**
         * 打印 HTTP 头字段
         * @param http
         */
        public static void printResponseHeader(HttpURLConnection http){
            Map<String, String> header = getHttpResponseHeader(http);
            for(Map.Entry<String, String> entry : header.entrySet()){
                 String key = entry.getKey()!=null ? entry.getKey()+ ":" : "";
                 print(key+ entry.getValue());
                 }
        }
        private static void print(String msg){
                Log.i(TAG, msg);
        }
}
```

(7) 编写文件 DownloadProgressListener.java，具体代码如下：

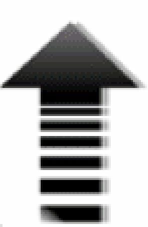
```
package com.changcheng.net.download;
public interface DownloadProgressListener {
        public void onDownloadSize(int size);
}
```

(8) 编写文件 FileService.java，具体代码如下：

```
package com.changcheng.download.service;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
/**
* 业务 bean
*
*/
public class FileService {
        private DBOpenHelper openHelper;
        public FileService(Context context) {
                openHelper = new DBOpenHelper(context);
        }
        /**
         * 获取线程最后下载位置
         * @param path
         * @return
         */
        public Map<Integer, Integer> getData(String path){
```




```
            SQLiteDatabase db = openHelper.getReadableDatabase();
            Cursor cursor = db.rawQuery("select threadid, position
               from filedown where downpath=?", new String[]{path});
            Map<Integer, Integer> data = new HashMap<Integer, Integer>();
            while(cursor.moveToNext()){
                    data.put(cursor.getInt(0), cursor.getInt(1));
            }
            cursor.close();
            db.close();
            return data;
    }
    /**
     * 保存下载线程初始位置
     * @param path
     * @param map
     */
    public void save(String path,  Map<Integer, Integer> map){//int
        threadid, int position
            SQLiteDatabase db = openHelper.getWritableDatabase();
            db.beginTransaction();
            try{
                 for(Map.Entry<Integer, Integer> entry : map.entrySet()){
                    db.execSQL("insert into filedown(downpath, threadid,
                        position) values(?,?,?)",
                    new Object[]{path, entry.getKey(), entry.getValue()});
                }
                    db.setTransactionSuccessful();
              }finally{
                    db.endTransaction();
            }
            db.close();
    }
    /**
     * 实时更新线程的最后下载位置
     * @param path
     * @param map
     */
    public void update(String path, Map<Integer, Integer> map){
        SQLiteDatabase db = openHelper.getWritableDatabase();
        db.beginTransaction();
        try{
            for(Map.Entry<Integer, Integer> entry : map.entrySet()){
            db.execSQL("update filedown set position=? where downpath=?
                and threadid=?",
            new Object[]{entry.getValue(), path, entry.getKey()});
            }
                db.setTransactionSuccessful();
            }finally{
                db.endTransaction();
            }
                db.close();
```



```
        }
        /**
         * 当文件下载完成后，清空该文件对应的下载记录
         * @param path
         */
        public void delete(String path){
            SQLiteDatabase db = openHelper.getWritableDatabase();
            db.execSQL("delete from filedown where downpath=?", new
                Object[]{path});
            db.close();
        }

}
```

(9) 编写文件 DownloadThread.java，具体代码如下：

```
package com.changcheng.net.download;
import java.io.InputStream;
import java.io.RandomAccessFile;
import java.net.HttpURLConnection;
import java.net.URL;
import android.util.Log;
public class DownloadThread extends Thread {
        private static final String TAG = "DownloadThread";
        private RandomAccessFile saveFile;
        private URL downUrl;
        private int block;
        /* 下载开始位置  */
        private int threadId = -1;
        private int startPos;
        private int downLength;
        private boolean finish = false;
        private FileDownloader downloader;
        public DownloadThread(FileDownloader downloader, URL downUrl,
            RandomAccessFile saveFile, int block, int startPos, int threadId) {
                this.downUrl = downUrl;
                this.saveFile = saveFile;
                this.block = block;
                this.startPos = startPos;
                this.downloader = downloader;
                this.threadId = threadId;
                this.downLength = startPos - (block * (threadId - 1));
        }
        @Override
        public void run() {
            if(downLength < block){//未下载完成
                try {
                    HttpURLConnection http = (HttpURLConnection)
                        downUrl.openConnection();
                    http.setRequestMethod("GET");
                    http.setRequestProperty("Accept", "image/gif, image/jpeg,
```

```
                image/pjpeg, image/pjpeg, application/x-shockwave-
                flash, application/xaml+xml, application/vnd.ms-
                xpsdocument, application/x-ms-xbap, application/
                x-ms-application, application/vnd.ms-excel, application/
                vnd.ms-powerpoint, application/msword, */*");
            http.setRequestProperty("Accept-Language", "zh-CN");
            http.setRequestProperty("Referer", downUrl.toString());
            http.setRequestProperty("Charset", "UTF-8");

            http.setRequestProperty("Range", "bytes=" +
                this.startPos + "-");
            http.setRequestProperty("User-Agent", "Mozilla/4.0
                (compatible; MSIE 8.0; Windows NT 5.2; Trident/4.0;
                .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
                3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR
                3.5.30729)");
            http.setRequestProperty("Connection", "Keep-Alive");
            InputStream inStream = http.getInputStream();
            int max = block>1024 ? 1024 : (block>10 ? 10 : 1);
            byte[] buffer = new byte[max];
            int offset = 0;
            print("线程 " + this.threadId + "从位置"+ this.startPos+
                "开始下载 ");

            while (downLength < block && (offset = inStream.read
                 (buffer, 0, max)) != -1) {
            saveFile.write(buffer, 0, offset);
            downLength += offset;
            downloader.update(this.threadId, block * (threadId - 1)
                + downLength);
            downloader.saveLogFile();
            downloader.append(offset);
            int spare = block-downLength;//求剩下的字节数
            if(spare < max) max = (int) spare;
        }
        saveFile.close();
        inStream.close();
        print("线程 " + this.threadId + "完成下载 ");
        this.finish = true;
        this.interrupt();
        } catch (Exception e) {
        this.downLength = -1;
        print("线程"+ this.threadId+ ":"+ e);
        }
    }
}
private static void print(String msg){
        Log.i(TAG, msg);
}
/**
 * 下载是否完成
```

```
         * @return
         */
        public boolean isFinish() {
                return finish;
        }
        /**
         * 已经下载的内容大小
         * @return 如果返回值为-1，代表下载失败
         */
        public long getDownLength() {
                return downLength;
        }
}
```

(10) 编写文件 DBOpenHelper.java，具体代码如下：

```
package com.changcheng.download.service;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
public class DBOpenHelper extends SQLiteOpenHelper {
        private static final String DBNAME = "download.db";
        private static final int VERSION = 2;
        public DBOpenHelper(Context context) {
                super(context, DBNAME, null, VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
           db.execSQL("CREATE TABLE IF NOT EXISTS filedown (id integer
           primary key autoincrement, downpath varchar(100), threadid
           INTEGER, position INTEGER)");
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                db.execSQL("DROP TABLE IF EXISTS filedown");
                onCreate(db);
        }
}
```

15.4　判断当前网络 GPRS 和 Wi-Fi 的状态

在 Android 开发过程中，特别是在开发和网络相关的一些应用时，很可能会用到网络连接状态，包括 GPRS、Wi-Fi 等。其实解决这些问题的方法很简单，Android 提供了两个类，一个是 ConnectivityManager，另一个是 NetworkInfo。通过这两个类即可判断当前网络 GPRS 和 Wi-Fi 的状态。

15.4.1 ConnectivityManager类和NetworkInfo类

在 Android 开发过程中，通过类 ConnectivityManager 可以实现管理和网络连接相关的操作，例如相关的 TelephonyManager 可以管理和手机、运营商等的相关信息，而 WifiManager 则管理和 Wi-Fi 相关的信息。

要想访问网络状态，首先得添加如下权限：

```
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

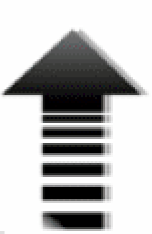
类 NetworkInfo 包含了对 Wi-Fi 和 Mobile 两种网络模式连接的详细描述，通过其 getState()方法获取的 State 对象则代表着连接成功与否等状态。

例如下面的代码演示了这两个类的基本用法。

```
  */
 public void testConnectivityManager() {
  ConnectivityManager connManager = (ConnectivityManager) this
    .getSystemService(CONNECTIVITY_SERVICE);
  // 获取代表联网状态的 NetWorkInfo 对象
  NetworkInfo networkInfo = connManager.getActiveNetworkInfo();
  // 获取当前的网络连接是否可用
  boolean available = networkInfo.isAvailable();
  if(available){
   Log.i("通知", "当前的网络连接可用");
  }
  else{
   Log.i("通知", "当前的网络连接可用");
  }
  State state = connManager.getNetworkInfo(ConnectivityManager.
    TYPE_MOBILE).getState();
  if(State.CONNECTED==state){
   Log.i("通知", "GPRS 网络已连接");
  }
  state =
connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI).getState();
  if(State.CONNECTED==state){
   Log.i("通知", "WIFI 网络已连接");
  }
  // 跳转到无线网络设置界面
  startActivity(new
Intent(android.provider.Settings.ACTION_WIRELESS_SETTINGS));
  // 跳转到无线 Wi-Fi 网络设置界面
  startActivity(new Intent(android.provider.Settings.ACTION_WIFI_SETTINGS));
}
```

了解了类 ConnectivityManager 和类 NetworkInfo 的基本用法后，判断当前网络 GPRS 和 Wi-Fi 状态的问题就迎刃而解了。例如通过下面的代码就可以实现。

```
import android.content.Context;
import android.net.ConnectivityManager;
```



```
import android.net.NetworkInfo;
public class Test extends Activity {
    private ConnectivityManager cm;
    private NetworkInfo info;
/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    cm = (ConnectivityManager)getSystemService
         (Context.CONNECTIVITY SERVICE);
    if (cm.getNetworkInfo(1).getState() == NetworkInfo.State.CONNECTED)
    {
        text.setText("WIFI 已经连接");
    }
    else {
        text.setText("WIFI 未连接");
    }
    if (cm.getNetworkInfo(0).getState() == NetworkInfo.State.CONNECTED)
    {

        text.setText("GPRS 已经连接");
    }
    else {
        text.setText("GPRS 未连接");
    }
}
}
```

通过下面的代码可以判断是否有网络连接。

```
    public boolean isNetworkConnected(Context context) {
        if (context != null) {
            ConnectivityManager mConnectivityManager = (ConnectivityManager)
                context
                    .getSystemService(Context.CONNECTIVITY SERVICE);
            NetworkInfo mNetworkInfo =
                mConnectivityManager.getActiveNetworkInfo();
            if (mNetworkInfo != null) {
                return mNetworkInfo.isAvailable();
            }
        }
        return false;
    }
```

通过下面的代码可以判断 Wi-Fi 网络是否可用。

```
    public boolean isWifiConnected(Context context) {
        if (context != null) {
            ConnectivityManager mConnectivityManager = (ConnectivityManager)
                context.getSystemService(Context.CONNECTIVITY SERVICE);
            NetworkInfo mWiFiNetworkInfo = mConnectivityManager
                    .getNetworkInfo(ConnectivityManager.TYPE_WIFI);
```

```
            if (mWiFiNetworkInfo != null) {
                return mWiFiNetworkInfo.isAvailable();
            }
        }
        return false;
    }
```

通过下面的代码可以判断 Mobile 网络是否可用。

```
    public boolean isMobileConnected(Context context) {
        if (context != null) {
            ConnectivityManager mConnectivityManager = (ConnectivityManager)
                context.getSystemService(Context.CONNECTIVITY SERVICE);
            NetworkInfo mMobileNetworkInfo = mConnectivityManager
                    .getNetworkInfo(ConnectivityManager.TYPE MOBILE);
            if (mMobileNetworkInfo != null) {
                return mMobileNetworkInfo.isAvailable();
            }
        }
        return false;
    }
```

通过下面的代码可以获取当前网络连接的类型信息。

```
    public static int getConnectedType(Context context) {
        if (context != null) {
            ConnectivityManager mConnectivityManager = (ConnectivityManager)
                context.getSystemService(Context.CONNECTIVITY SERVICE);
            NetworkInfo mNetworkInfo =
                 mConnectivityManager.getActiveNetworkInfo();
            if (mNetworkInfo != null && mNetworkInfo.isAvailable()) {
                return mNetworkInfo.getType();
            }
        }
        return -1;
    }
```

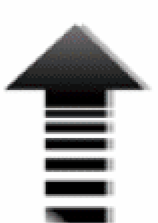
15.4.2 在程序启动时对网络状态进行判断

在使用 Android 连接网络的时候，并不是每次都能连接到网络，这时最好在程序启动时对网络的状态进行判断，如果没有网络则即时提醒用户进行设置。要判断网络状态，首先需要有相应的权限。下面为允许访问网络状态权限的代码：

```
<uses-permission android:name="android.permission.ACCESS NETWORK STATE">
</uses-permission>
```

下面为具体的判断代码：

```
  private boolean NetWorkStatus() {
          boolean netSataus = false;
          ConnectivityManager cwjManager = (ConnectivityManager)
```



```
        getSystemService(Context.CONNECTIVITY SERVICE);
    cwjManager.getActiveNetworkInfo();
    if (cwjManager.getActiveNetworkInfo() != null) {
        netSataus = cwjManager.getActiveNetworkInfo().isAvailable();
    }
    if (netSataus) {
        Builder b = new AlertDialog.Builder(this).setTitle("没有可用
            的网络")
                .setMessage("是否对网络进行设置？");
        b.setPositiveButton("是", new DialogInterface.
              OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                Intent mIntent = new Intent("/");
                ComponentName comp = new ComponentName(
                        "com.android.settings",
                        "com.android.settings.WirelessSettings");
                mIntent.setComponent(comp);
                mIntent.setAction("android.intent.action.VIEW");
          // 如果在设置完成后需要再次进行操作，可以重写操作代码，在这里不再重写
                startActivityForResult(mIntent,0);
            }
        }).setNeutralButton("否", new DialogInterface.OnClickListener() {
             public void onClick(DialogInterface dialog, int whichButton) {
                dialog.cancel();
            }
        }).show();
    }
    return netSataus;
}
```

15.5　开启或关闭 APN

虽然 Android 对于 APN 的网络 API 没有公开，但是我们可以参考源代码，然后进行数据库操作，这样系统会自动监听数据库的变化，从而实现开启或者关闭 APN 的操作。读者在获取 Android 的源代码后，可以重点研究一下文件 frameworks/base/core/Java/android/provider/Telephony.java 中的类。此类的核心是 URI 和数据库字段 content://telephony/carriers，该字段可以在文件 Telephony.java 中找到。

下面的演示代码实现了如下两个功能。

(1) 当开启 APN 的时候，设置一个正确的移动或者联通的 APN。

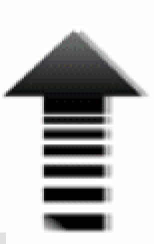
(2) 关闭的时候设置一个错误的 APN 就会自动关闭网络。

首先定义继承于 Activity 的类 Main，代码如下：

```
package cc.mdev.Demo;
import java.util.ArrayList;
import java.util.List;
```

```
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
/**
* Activity
* @author SinFrancis wong
*/
public class Main extends Activity {
/** Called when the activity is first created. */
Uri uri = Uri.parse("content://telephony/carriers");
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
Button open= (Button) findViewById(R.id.open);
Button close= (Button) findViewById(R.id.close);
open.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
openAPN();
}
});
close.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {

closeAPN();
}
});
}
public void openAPN(){
List list = getAPNList();
for (APN apn : list) {
ContentValues cv = new ContentValues();
cv.put("apn", APNMatchTools.matchAPN(apn.apn));
cv.put("type", APNMatchTools.matchAPN(apn.type));
getContentResolver().update(uri, cv, " id=?", new String[]{apn.id});
}
}
public void closeAPN(){
List list = getAPNList();
for (APN apn : list) {
ContentValues cv = new ContentValues();
cv.put("apn", APNMatchTools.matchAPN(apn.apn)+"mdev");
cv.put("type", APNMatchTools.matchAPN(apn.type)+"mdev");
getContentResolver().update(uri, cv, "_id=?", new String[]{apn.id});
```



```
}
}
private List getAPNList(){
String tag = "Main.getAPNList()";
//current 不为空表示可以使用的 APN
String projection[] = {" id,apn,type,current"};
Cursor cr = this.getContentResolver().query(uri, projection, null,
   null, null);
List list = new ArrayList();
while(cr!=null && cr.moveToNext()){
Log.d(tag, cr.getString(cr.getColumnIndex(" id")) + " " +
      cr.getString(cr.getColumnIndex("apn")) + " " +
      cr.getString(cr.getColumnIndex("type"))+ " " +
      cr.getString(cr.getColumnIndex("current")));
APN a = new APN();
a.id = cr.getString(cr.getColumnIndex(" id"));
a.apn = cr.getString(cr.getColumnIndex("apn"));
a.type = cr.getString(cr.getColumnIndex("type"));
list.add(a);
}
if(cr!=null)
cr.close();
return list;
}
public static class APN{
String id;
String apn;
String type;
}
}
```

然后实现不同运营商的 APN，演示代码如下：

```
package cc.mdev.apn;
public final class APNMatchTools {
public static class APNNet{
/*
* 中国移动 cmwap
*/
public static String CMWAP = "cmwap";
/**
* 中国移动 cmnet
*/
public static String CMNET = "cmnet";

//中国联通 3GWAP 设置 中国联通 3G 因特网设置 中国联通 WAP 设置 中国联通因特网设置
//3gwap 3gnet uniwap uninet
/**
* 3G wap 中国联通 3gwap APN
*/
public static String GWAP_3 = "3gwap";
```

```
/**
* 3G net 中国联通 3gnet APN
*/
public static String GNET 3="3gnet";
/**
* uni wap 中国联通 uni wap APN
*/
public static String UNIWAP="uniwap";
/**
* uni net 中国联通 uni net APN
*/

public static String UNINET="uninet";
}
public static String matchAPN(String currentName) {
if("".equals(currentName) || null==currentName){
return "";
}
currentName = currentName.toLowerCase();
if(currentName.startsWith(APNNet.CMNET))
return APNNet.CMNET;
else if(currentName.startsWith(APNNet.CMWAP))
return APNNet.CMWAP;
else if(currentName.startsWith(APNNet.GNET 3))
return APNNet.GNET 3;
else if(currentName.startsWith(APNNet.GWAP 3))
return APNNet.GWAP 3;
else if(currentName.startsWith(APNNet.UNINET))
return APNNet.UNINET;
else if(currentName.startsWith(APNNet.UNIWAP))
return APNNet.UNIWAP;
else if(currentName.startsWith("default"))
return "default";
else return "";
// return currentName.substring(0, currentName.length() - SUFFIX.length());
}
}
```

Android

第16章 开发一个邮件系统

现代社会科技发展迅速，移动设备在我们生活中应用广泛，主要用于娱乐、办公、科研等多个领域。基于移动设备上的应用系统发展也相当迅速，同时移动设备运用领域的扩大也提高了对系统的要求。Android 作为一个开源系统，为移动设备市场的发展提供了机遇。本章的邮件系统实例采用 Android 开源系统技术，利用 Java 语言和 Eclipse 开发工具对邮件系统进行开发，同时给出详细的系统设计流程、部分界面图及主要功能效果流程图。在本章的内容中，还对开发过程中遇到的问题和解决方法进行详细的讨论。邮件系统实例集用户设置、邮件收取和邮件发送等功能于一体，在 Android 系统中能独立运行。

16.1 项 目 介 绍

本章邮件系统源码保存在本书附带光盘中的“光盘:\daima\16”目录下。在讲解具体编码之前，先简要介绍本项目的产生背景和项目意义，为后面的系统设计及编码做准备。

16.1.1 项目背景

随着科学技术的发展，计算机进入了生活，娱乐和办公。在计算机系统中，使用邮件系统收发邮件是工作中必不可少的组成部分，如今社会竞争十分激烈，工作效率显得越发重要。使用手机或者是便捷设备在旅行、出差或者是路上处理工作事务和朋友间的联系越来越流行。因此，人们的生活越来越离不开手机的陪伴。随着手机硬件和软件系统的发展，人们对移动电子设备的硬件性能和软件性能要求也越来越高。

全球最大的移动设备手机发展十分迅速，同时手机操作系统也出现了不同种类，目前市场上主要有三种手机操作系统：微软的 Windows Phone、苹果的 ISO 和 Google 的 Android 操作系统，其中只有 Android 开放源代码。全球针对 Android 平台开发的团体和个人数量庞大，因此 Android 系统得以飞速发展。既然手机如此智能，我们通过手机接收邮件可以实现吗？答案是肯定的！谷歌的 Android 系统可以满足你的要求。本章讲解的邮件系统实例就是基于谷歌的 Android 手机平台开发的。

开发一个邮件系统，要了解邮件系统支持的通信协议，以及各协议之间存在哪些差异，还要对开发平台有较深入的了解，分析现在流行的邮件系统中的优点、缺点和用户最常用的功能。

16.1.2 项目目的

目前社会竞争激烈，提高工作效率越来越重要，而互联网办公是其中最好的提高工作效率的方式之一。本项目的目的是开发一个在手机或者是移动设备上使用的邮件系统，该系统的主要功能是邮箱类型设定、邮件收取设置、邮件发送设置、用户检查、用户别名设置和编辑邮件，支持 POP3 和 IMAP 通信协议，检查用户的设定是否正确。系统界面简明，操作简单。

本项目是基于 Android 手机平台的邮件系统，让 Android 手机拥有个性的邮件系统，使手机显得更方便和智能，与人们更为亲近，手机主人可以随时随地处理工作事务或者是与朋友联系，使人们的生活更加多样化，也使设计者更加熟练 Android 的技术和其他在市场上的特点。

16.2　系统需求分析

根据项目的目标，我们可分析出系统的基本需求。以下从软件设计的角度来描述系统的功能，并且使用例图来描述系统的功能模块，大致分成五部分来概括，即邮箱类型设置、邮箱收取设置、邮箱发送设置、邮箱用户检查和用户邮件编辑。

16.2.1　构成模块

邮件系统的构成模块如图 16-1 所示。

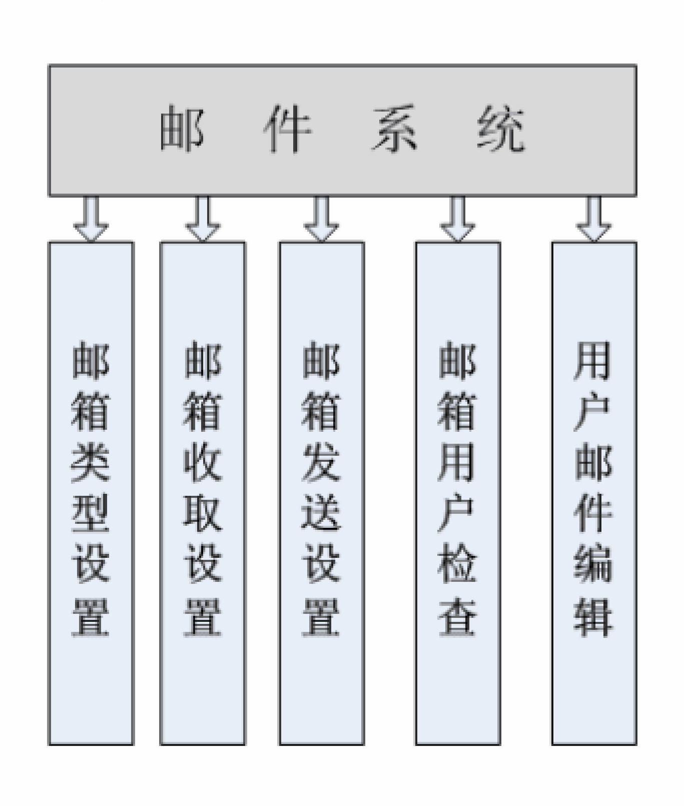


图 16-1　邮件系统构成模块

邮件系统各个模块的具体说明如下。

1. 邮箱类型设置

此模块的功能是设置通信协议。

1) POP3 协议

- 目标：使得用户可以收发邮件到本地。
- 前置条件：成功登录邮件系统。
- 基本事件流：
  - 用户单击 Next 按钮。
  - 程序进入邮箱收取设置。

2) IMAP 协议

- 目标：使得用户可以在线收发邮件。
- 前置条件：成功登录邮件系统。
- 基本事件流：
  - 用户单击 Next 按钮。
  - 程序进入邮箱收取设置。

邮箱类型设置界面结构如图 16-2 所示。

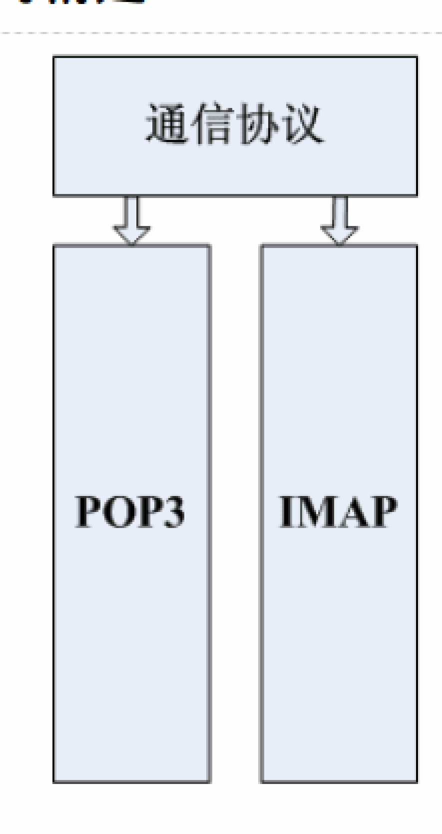


图 16-2　邮箱类型设置界面结构

2. 邮箱收取设置

当用户选定通信协议后，可以进行邮箱收取功能的设置。

- 目标：设定用户基本信息。
- 前置条件：程序运行在用户基本信息设定界面。
- 基本事件流：
  - 用户填写用户名和密码。
  - 用户填写服务器名和端口。
  - 用户填写加密协议。
  - 用户设定邮件删除期限。
  - 用户单击 Next 按钮。

邮箱收取设置界面结构如图 16-3 所示。

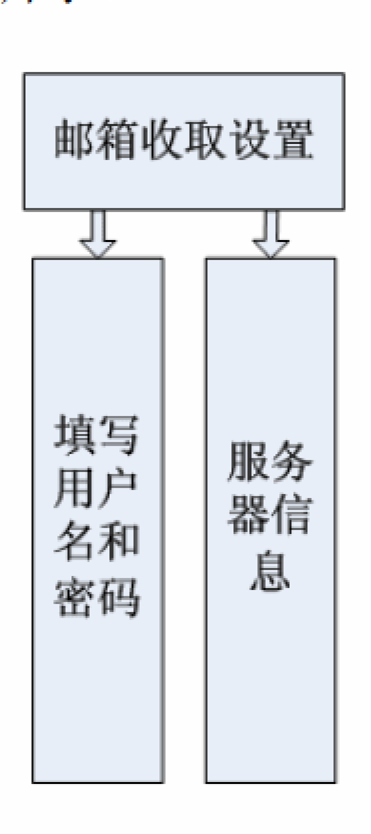


图 16-3　邮箱收取设置界面结构

3. 邮箱发送设置

本模块用于设置邮箱发送。

- 目标：设定邮箱发送。
- 前置条件：程序运行在邮箱发送设定界面。
- 基本事件流：

◆ 用户填写服务器名和端口。
◆ 用户单击 Next 按钮。

邮箱发送设置界面结构如图 16-4 所示。

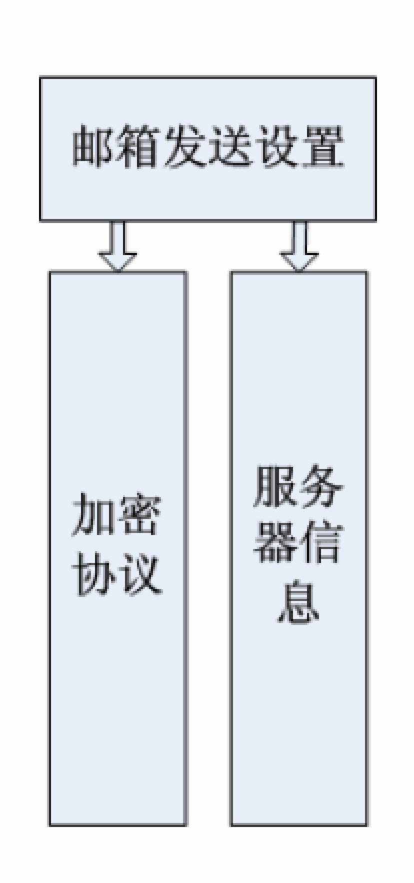


图 16-4　邮箱发送设置界面结构

4. 邮箱用户检查

此模块的功能是邮箱用户检查。

1) 用户名和密码验证

- 目标：验证用户名和密码的正确性。
- 前置条件：程序运行在主界面。
- 基本事件流。

2) 接收地址验证

- 目标：验证接收地址的正确性。
- 前置条件：程序运行在目录界面。
- 基本事件流。

3) 发送地址验证

- 目标：验证发送地址的正确性。
- 前置条件：程序运行在目录界面。
- 基本事件流：用户单击 Next 按钮。

邮箱用户检查界面结构如图 16-5 所示。

5. 用户邮件编辑

此模块的功能是用户邮件编辑。

- 目标：编辑邮件。
- 前置条件：进入邮件编辑界面。
- 基本事件流：
  - ◆ 用户填写收件人地址。
  - ◆ 用户填写标题。
  - ◆ 用户填写邮件内容。

◆ 用户单击 Send 按钮。

用户邮件编辑界面结构如图 16-6 所示。

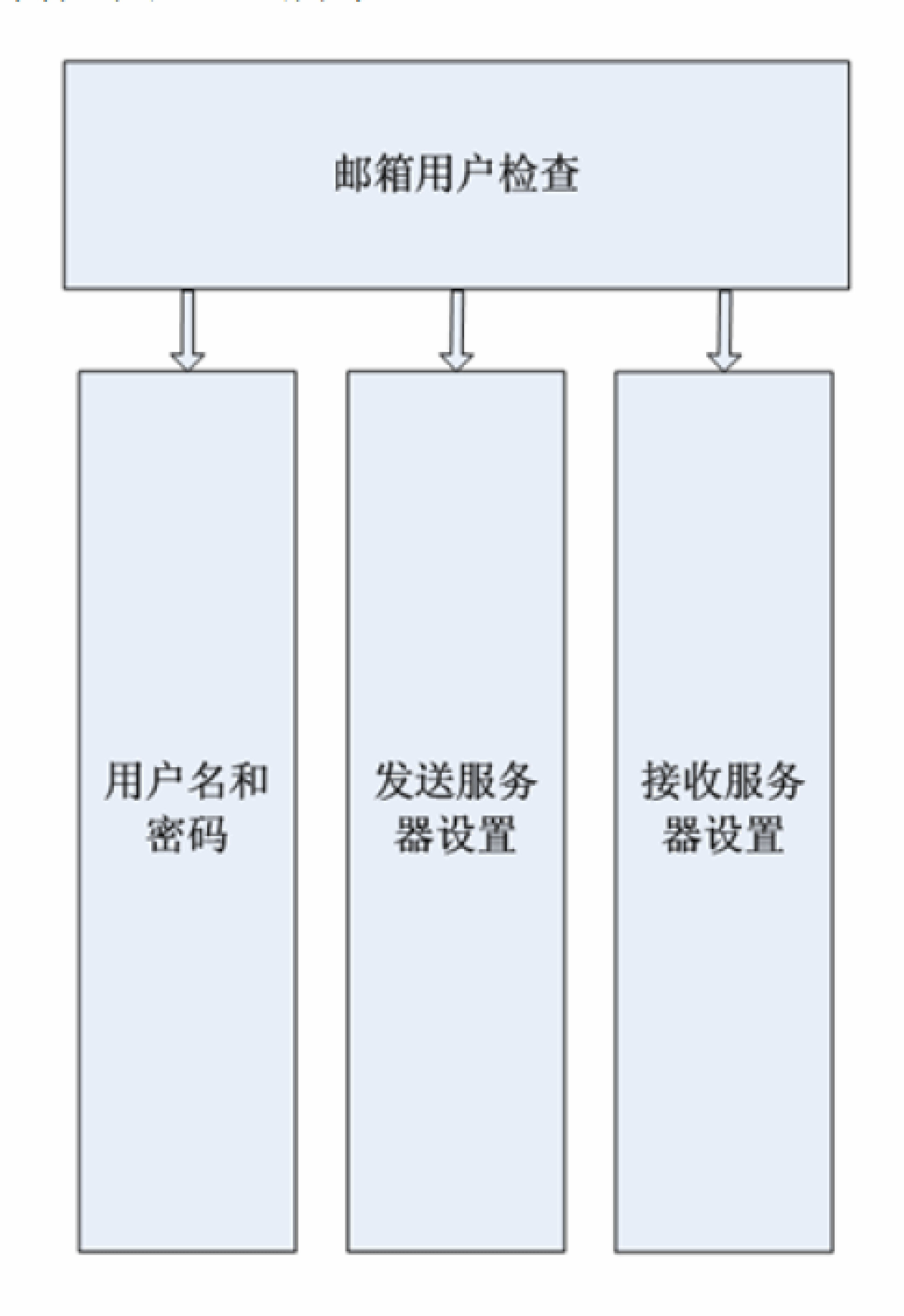


图 16-5 邮箱用户检查界面结构

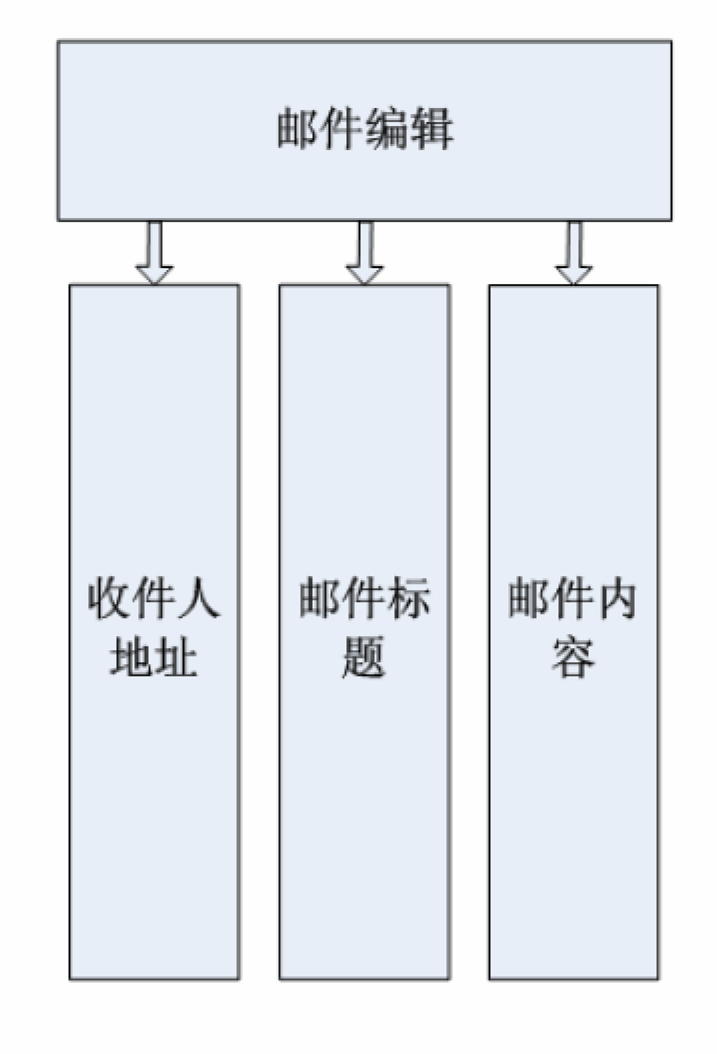


图 16-6 邮件编辑界面结构

16.2.2 系统流程

邮件系统流程如图 16-7 所示。



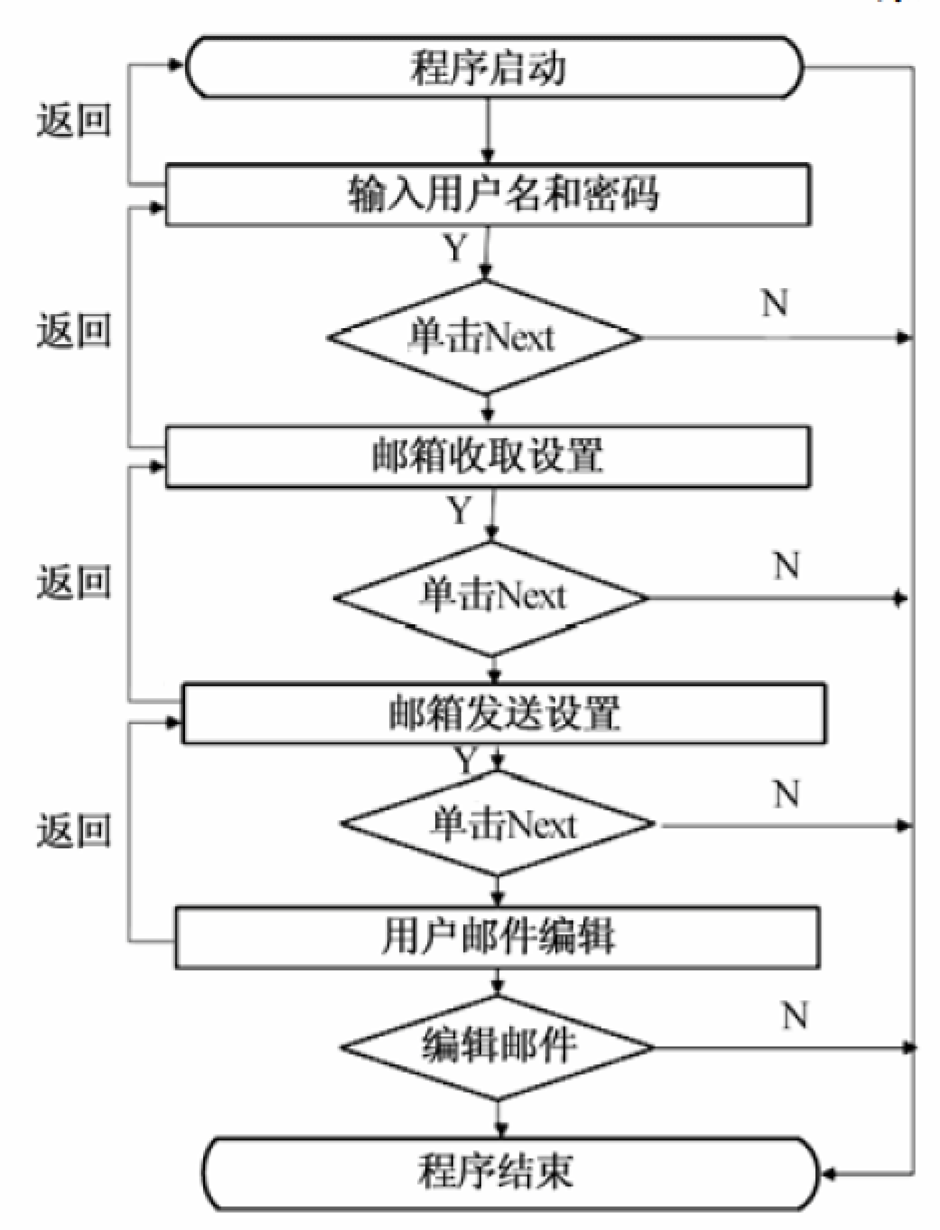


图 16-7　邮件系统流程图

16.2.3　功能结构图

邮件系统的完整功能结构如图 16-8 所示。

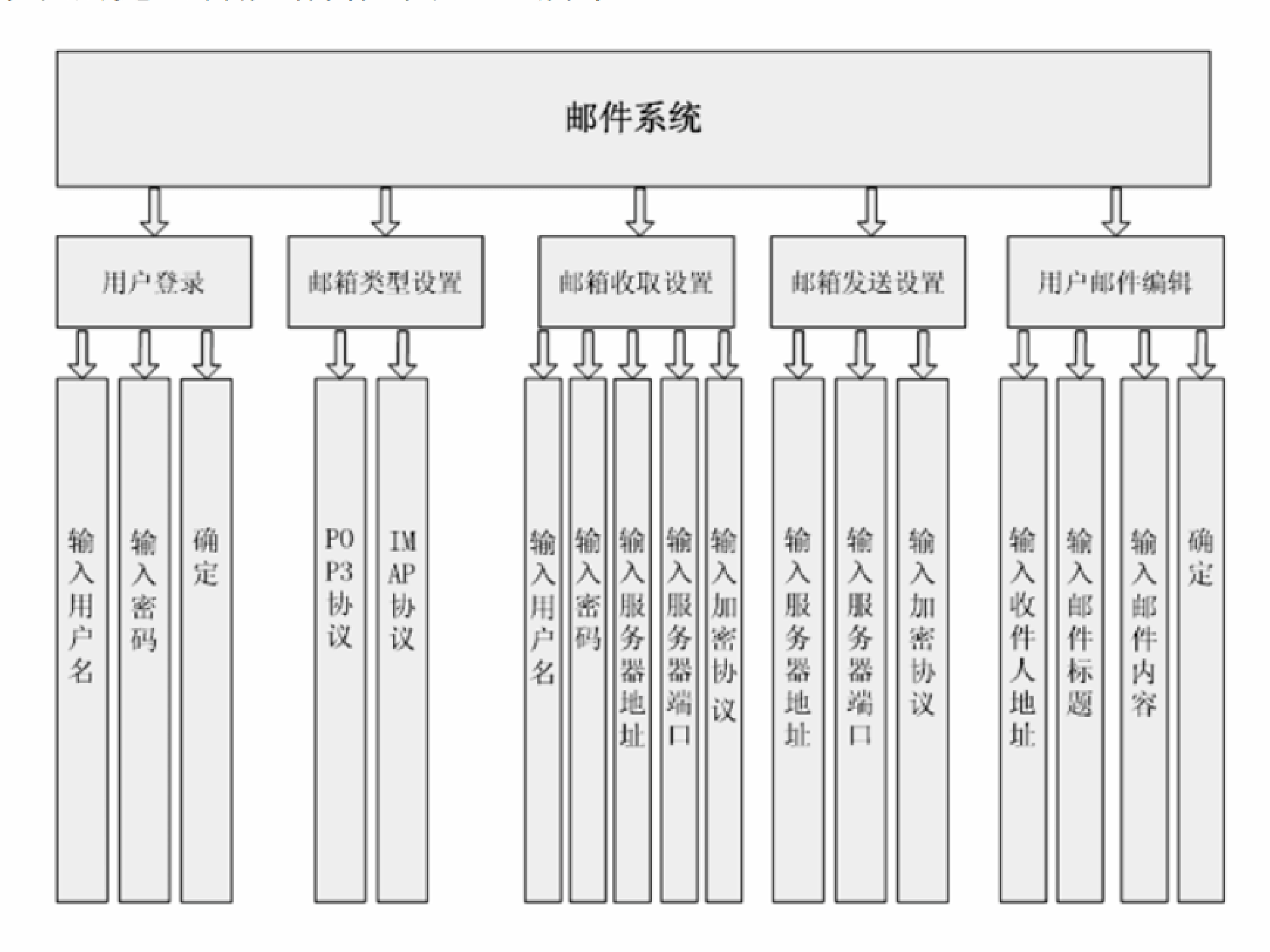


图 16-8　邮件系统完整功能结构图

16.2.4　系统功能说明

邮件系统各个模块功能的说明如表 16-1 所示。

表 16-1　模块结构功能说明信息

功能类别	子 功 能
用户登录	输入用户名
	输入密码
	进入邮件登录设置
邮箱类型设置	POP3 协议
	IMAP 协议
邮箱用户检查模块	
邮箱收取设置	输入用户名
	输入密码
	输入服务器地址
	输入服务器端口
	输入加密协议
邮箱发送设置	输入服务器地址
	输入服务器端口
	输入加密协议
用户邮件编辑	输入收件人地址
	输入邮件标题
	输入邮件内容
	单击“发送”按钮

16.2.5　系统需求

1. 系统性能需求

根据 Android 手机系统要求无响应时间为 5 秒，所以就有以下性能要求。

- 邮箱类型设置，程序响应时间最长不能超过 5 秒。
- 邮箱收取设置，程序响应时间最长不能超过 5 秒。
- 邮箱发送设置，程序响应时间最长不能超过 5 秒。
- 邮箱用户检查，程序响应时间最长不能超过 5 秒。
- 用户邮件编辑，程序响应时间最长不能超过 5 秒。

2. 运行环境需求

- 操作系统：Android 手机基于 Linux 操作系统。
- 支持环境：Android 1.5 - 2.0.1 版本。
- 开发环境：Eclipse 3.5 ADT 0.95。



16.3 数据存储设计

基于 Windows 或者是 Linux 的大型系统开发，数据库使用的都是专业数据库系统，Android 开发平台提供了几种数据存储：Android 系统中自带的 iSQLite 数据库、数据接口共享数据(SharedPreferences)模式保存数据、文件方式保存数据、内容提供器(Contextprovider)和网络方式保存数据。数据库是存放数据的仓库，只不过这个仓库是在计算机存储设备上，而且数据是按一定的格式存放的。本实例采用 SharedPreferences 保存数据。

SharedPreferences 是以 XML 格式文件的方式自动保存，在 DDMS 的 File Exploer 下展开/data/data/<package name>/shared_prefs，生成 AndroidMail.Main.xml 文件。

16.3.1 用户信息类

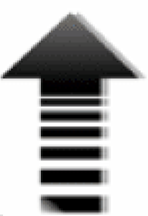
定义用户信息 Account.java 类，此类将保存系统用户有关的所有信息。为 SharedPreferences 模式保存数据提供用户实例对象。对应代码如下：

```
public class Account implements Serializable {
public static final int DELETE POLICY NEVER = 0;
public static final int DELETE POLICY 7DAYS = 1;
public static final int DELETE POLICY ON DELETE = 2;
private static final long serialVersionUID = 2975156672298625121L;
String mUuid;                                              //邮件用户 ID
String mStoreUri;                                          //邮件源地址
String mLocalStoreUri;
String mSenderUri;                                         //邮件目的地址
String mDescription;                                       //邮件内容
String mName;                                              //用户名
String mEmail;
int mAutomaticCheckIntervalMinutes;
long mLastAutomaticCheckTime;
boolean mNotifyNewMail;
String mDraftsFolderName;
String mSentFolderName;
String mTrashFolderName;
String mOutboxFolderName;
int mAccountNumber;
boolean mVibrate;
String mRingtoneUri;
int mDeletePolicy;
  //初始化
public Account(Context context) {
    mUuid = UUID.randomUUID().toString();
    mLocalStoreUri = "local://localhost/" + context.getDatabasePath
         (mUuid + ".db");
    mAutomaticCheckIntervalMinutes = -1;
    mAccountNumber = -1;
```

```
        mNotifyNewMail = true;
        mVibrate = false;
        mRingtoneUri = "content://settings/system/notification sound";
    }
   //刷新指定用户
    Account(Preferences preferences, String uuid) {
        this.mUuid = uuid;
        refresh(preferences);
    }
     //刷新
    public void refresh(Preferences preferences) {
        mStoreUri = Utility.base64Decode(preferences.mSharedPreferences.getString
               (mUuid + ".storeUri", null));
```

(1) 通过 SharedPreferences 对象的 getString()方法取得保存在其中的值。对应代码如下：

```
        mLocalStoreUri = preferences.mSharedPreferences.getString(mUuid +
            ".localStoreUri", null);
        String senderText = preferences.mSharedPreferences.getString(mUuid
            + ".senderUri", null);
        if (senderText == null) {
            //获取 ID
            senderText = preferences.mSharedPreferences.getString(mUuid +
                ".transportUri", null);
        }
       //转换编码方式
        mSenderUri = Utility.base64Decode(senderText);
        mDescription = preferences.mSharedPreferences.getString(mUuid +
            ".description", null);
       //获取与此用户身份有关的信息
        mName = preferences.mSharedPreferences.getString(mUuid + ".name",
            mName);
        mEmail = preferences.mSharedPreferences.getString(mUuid + ".email",
            mEmail);
        mAutomaticCheckIntervalMinutes = preferences.mSharedPreferences.getInt
               (mUuid + ".automaticCheckIntervalMinutes", -1);
        mLastAutomaticCheckTime = preferences.mSharedPreferences.getLong
               (mUuid + ".lastAutomaticCheckTime", 0);
        mNotifyNewMail = preferences.mSharedPreferences.getBoolean
               (mUuid + ".notifyNewMail", false);
        mDraftsFolderName = preferences.mSharedPreferences.getString
               (mUuid + ".draftsFolderName", "Drafts");
        mSentFolderName = preferences.mSharedPreferences.getString
               (mUuid + ".sentFolderName", "Sent");
        mTrashFolderName = preferences.mSharedPreferences.getString
               (mUuid  + ".trashFolderName", "Trash");
        mOutboxFolderName = preferences.mSharedPreferences.getString
               (mUuid  + ".outboxFolderName", "Outbox");
        mAccountNumber = preferences.mSharedPreferences.getInt
               (mUuid + ".accountNumber", 0);
        mVibrate = preferences.mSharedPreferences.getBoolean
```



```
            (mUuid + ".vibrate", false);
        mRingtoneUri = preferences.mSharedPreferences.getString
            (mUuid  + ".ringtone",
            "content://settings/system/notification sound");
    }
    public String getUuid() {
        return mUuid;
    }
    public String getStoreUri() {
        return mStoreUri;
    }
```

(2) 通过各属性的 set()方法赋值。对应代码如下：

```
    //为属性赋值
public void setStoreUri(String storeUri) {
        this.mStoreUri = storeUri;
    }
    public String getSenderUri() {
        return mSenderUri;
    }
    public void setSenderUri(String senderUri) {
        this.mSenderUri = senderUri;
    }
    public String getDescription() {
        return mDescription;
    }
    public void setDescription(String description) {
        this.mDescription = description;
    }
    public String getName() {
        return mName;
    }
    public void setName(String name) {
        this.mName = name;
    }
    public String getEmail() {
        return mEmail;
    }
    public void setEmail(String email) {
        this.mEmail = email;
    }
    public boolean isVibrate() {
        return mVibrate;
    }
    public void setVibrate(boolean vibrate) {
        mVibrate = vibrate;
    }
    public String getRingtone() {
        return mRingtoneUri;
    }
```

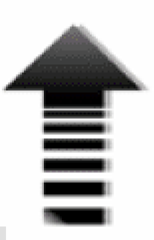
```
public void setRingtone(String ringtoneUri) {
        mRingtoneUri = ringtoneUri;
    }
```

(3) 定义 delete()方法删除指定的 Account 实例，对象 SharedPreferences.Editor 中的 Remove()方法执行删除值操作。commit()方法对所做的修改提交。对应代码如下：

```
public void delete(Preferences preferences) {
    String[] uuids = preferences.mSharedPreferences.getString
         ("accountUuids", "").split(",");
    StringBuffer sb = new StringBuffer();
    for (int i = 0, length = uuids.length; i < length; i++) {
        if (!uuids[i].equals(mUuid)) {
            if (sb.length() > 0) {
                sb.append(',');
            }
            sb.append(uuids[i]);
        }
    }
    String accountUuids = sb.toString();
   //定义 SharedPreferences.Editor 对象，对指定值的清除
    SharedPreferences.Editor editor = preferences.mSharedPreferences.edit();
    editor.putString("accountUuids", accountUuids);
    editor.remove(mUuid + ".storeUri");
    editor.remove(mUuid + ".localStoreUri");
    editor.remove(mUuid + ".senderUri");
    editor.remove(mUuid + ".description");
    editor.remove(mUuid + ".name");
    editor.remove(mUuid + ".email");
    editor.remove(mUuid + ".automaticCheckIntervalMinutes");
    editor.remove(mUuid + ".lastAutomaticCheckTime");
    editor.remove(mUuid + ".notifyNewMail");
    editor.remove(mUuid + ".deletePolicy");
    editor.remove(mUuid + ".draftsFolderName");
    editor.remove(mUuid + ".sentFolderName");
    editor.remove(mUuid + ".trashFolderName");
    editor.remove(mUuid + ".outboxFolderName");
    editor.remove(mUuid + ".accountNumber");
    editor.remove(mUuid + ".vibrate");
    editor.remove(mUuid + ".ringtone");
    editor.remove(mUuid + ".transportUri");
  // 提交所做的操作
    editor.commit();
}
```

(4) 定义 save()方法保存指定的 Account 实例，对象 SharedPreferences.Editor 中的方法 putString()保存指定的值，该方法的第一个参数是键名；第二个参数是值。对应代码如下：

```
public void save(Preferences preferences) {
    if (!preferences.mSharedPreferences.getString("accountUuids",
        "").contains(mUuid)) {
        Account[] accounts = preferences.getAccounts();
```



```
            int[] accountNumbers = new int[accounts.length];
            for (int i = 0; i < accounts.length; i++) {
                accountNumbers[i] = accounts[i].getAccountNumber();
            }
            Arrays.sort(accountNumbers);
            for (int accountNumber : accountNumbers) {
                if (accountNumber > mAccountNumber + 1) {
                    break;
                }
                mAccountNumber = accountNumber;
            }
            mAccountNumber++;
            String accountUuids = preferences.mSharedPreferences.getString
                 ("accountUuids", "");
            accountUuids += (accountUuids.length() != 0 ? "," : "") + mUuid;
            SharedPreferences.Editor editor = preferences.mSharedPreferences.edit();
           //  保存 accountUuids 名的值
            editor.putString("accountUuids", accountUuids);
         // 提交所做的操作
            editor.commit();
        }
        SharedPreferences.Editor editor = preferences.mSharedPreferences.edit();
        editor.putString(mUuid + ".storeUri", Utility.base64Encode(mStoreUri));
        editor.putString(mUuid + ".localStoreUri", mLocalStoreUri);
        editor.putString(mUuid + ".senderUri", Utility.base64Encode(mSenderUri));
        editor.putString(mUuid + ".description", mDescription);
        editor.putString(mUuid + ".name", mName);
        editor.putString(mUuid + ".email", mEmail);
        editor.putInt(mUuid + ".automaticCheckIntervalMinutes",
             mAutomaticCheckIntervalMinutes);
        editor.putLong(mUuid + ".lastAutomaticCheckTime", mLastAutomaticCheckTime);
        editor.putBoolean(mUuid + ".notifyNewMail", mNotifyNewMail);
        editor.putInt(mUuid + ".deletePolicy", mDeletePolicy);
        editor.putString(mUuid + ".draftsFolderName", mDraftsFolderName);
        editor.putString(mUuid + ".sentFolderName", mSentFolderName);
        editor.putString(mUuid + ".trashFolderName", mTrashFolderName);
        editor.putString(mUuid + ".outboxFolderName", mOutboxFolderName);
        //  保存整型数据
        editor.putInt(mUuid + ".accountNumber", mAccountNumber);
        //  保存逻辑型数据
        editor.putBoolean(mUuid + ".vibrate", mVibrate);
        editor.putString(mUuid + ".ringtone", mRingtoneUri);
        editor.remove(mUuid + ".transportUri");
        editor.commit();
    }
}
```

16.3.2　SharedPreferences

定义 Preferences.java 类，该类基于类 SharedPreferences。使用方法 getSharedPreferences()



返回 mSharedPreferences 对象，返回功能对应的代码如下：

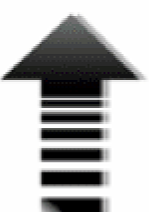
```
public class Preferences {
    private static Preferences preferences;
    SharedPreferences mSharedPreferences;
    private Preferences(Context context) {
        //读取数据
        mSharedPreferences = context.getSharedPreferences("AndroidMail.Main",
            Context.MODE PRIVATE);
    }
    public static synchronized Preferences getPreferences(Context context) {
        if (preferences == null) {
            preferences = new Preferences(context);
        }
        return preferences;
    }
```

(1) 定义 getAccounts()方法，返回 accountUuids 对应的值。对应代码如下：

```
    public Account[] getAccounts() {
        String accountUuids = mSharedPreferences.getString("accountUuids", null);
        if (accountUuids == null || accountUuids.length() == 0) {
            return new Account[] {};
        }
        String[] uuids = accountUuids.split(",");
        Account[] accounts = new Account[uuids.length];
        for (int i = 0, length = uuids.length; i < length; i++) {
            accounts[i] = new Account(this, uuids[i]);
        }
        return accounts;
    }
```

(2) 定义 getAccountByContentUri()方法，返回指定邮箱类型对应的 URL 值。对应代码如下：

```
    public Account getAccountByContentUri(Uri uri) {
        if (!"content".equals(uri.getScheme()) || !"accounts".equals
            (uri.getAuthority())) {
            return null;
        }
        String uuid = uri.getPath().substring(1);
        if (uuid == null) {
            return null;
        }
        String accountUuids = mSharedPreferences.getString("accountUuids", null);
        if (accountUuids == null || accountUuids.length() == 0) {
            return null;
        }
        String[] uuids = accountUuids.split(",");
        for (int i = 0, length = uuids.length; i < length; i++) {
            if (uuid.equals(uuids[i])) {
                return new Account(this, uuid);
```



```
            }
        }
        return null;
    }
```

(3) 定义 getDefaultAccount ()方法，返回默认的用户 ID。对应代码如下：

```
    public Account getDefaultAccount() {
        String defaultAccountUuid = mSharedPreferences.getString
             ("defaultAccountUuid", null);
        Account defaultAccount = null;
        Account[] accounts = getAccounts();
        if (defaultAccountUuid != null) {
            for (Account account : accounts) {
                if (account.getUuid().equals(defaultAccountUuid)) {
                    defaultAccount = account;
                    break;
                }
            }
        }
        if (defaultAccount == null) {
            if (accounts.length > 0) {
                defaultAccount = accounts[0];
                setDefaultAccount(defaultAccount);
            }
        }
        return defaultAccount;
    }
    public void setDefaultAccount(Account account) {
        mSharedPreferences.edit().putString("defaultAccountUuid",
           account.getUuid()).commit();
    }
```

(4) 定义 setEnableDebugLogging ()方法赋值是否开启调试信息，该方法读取调试信息开启情况，对应代码如下：

```
    public void setEnableDebugLogging(boolean value) {
        mSharedPreferences.edit().putBoolean("enableDebugLogging", value).commit();
    }
    public boolean geteEnableDebugLogging() {
        return mSharedPreferences.getBoolean("enableDebugLogging", false);
    }
  public void setEnableSensitiveLogging(boolean value) {
   //直接修改 enableSensitiveLogging 的值
        mSharedPreferences.edit().putBoolean("enableSensitiveLogging",
            value).commit();
    }
    public boolean getEnableSensitiveLogging() {
        return mSharedPreferences.getBoolean("enableSensitiveLogging", false);
    }
    public void save() {
    }
```

```
    public void clear() {
         //清除对象里的键名
        mSharedPreferences.edit().clear().commit();
    }
    public void dump() {
        if (Config.LOGV) {
            for (String key : mSharedPreferences.getAll().keySet()) {
                Log.v(Email.LOG TAG, key + " = " + mSharedPreferences.
                        getAll().get(key));
            }
        }
    }
}
```

16.4 具 体 编 码

经过前面内容的讲解，本项目的前期工作已经结束。在接下来的内容中，将详细讲解本项目的具体编码过程。

16.4.1 欢迎界面

欢迎界面是整个项目的入口，通过入口可进入到系统的其他功能。欢迎界面如图 16-9 所示。

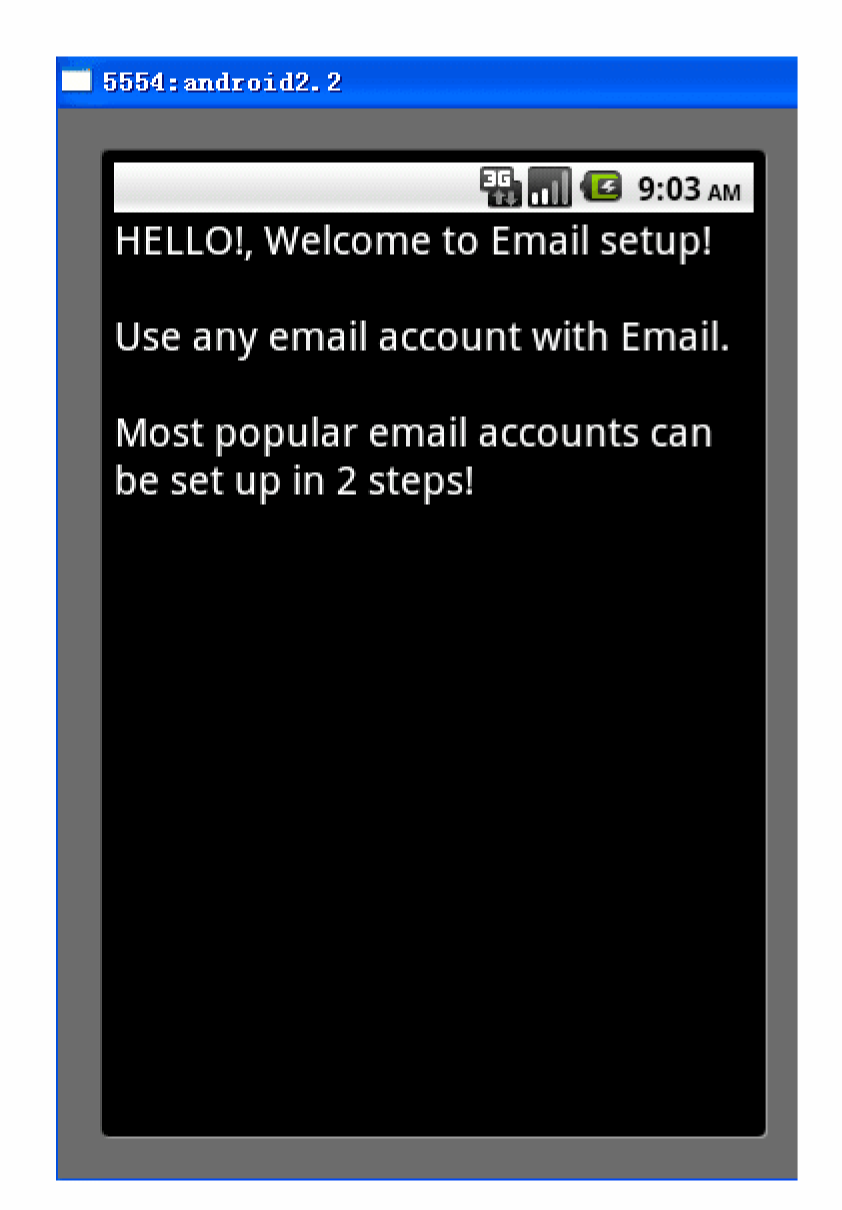


图 16-9 欢迎界面

(1) 编写文件 WelActivity.java，在此定义项目的欢迎界面，主要代码如下。

① 定义 WelActivity 类继承 ListActivity 类，ListActivity 类又继承 Activity。ListActivity



默认绑定了一个 ListView(列表视图)界面组件，提供一些与视图、处理相关的操作。

```
public class WelActivity extends ListActivity implements
OnItemClickListener, OnClickListener{
    private static final String EXTRA ACCOUNT = "account";
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 加载 activity wel.xml 布局
        setContentView(R.layout.activity wel);
        ListView listView = getListView();
        listView.setOnItemClickListener(this);
       listView.setItemsCanFocus(false);
      //获取指定的组件
       listView.setEmptyView(findViewById(R.id.empty));
       findViewById(R.id.add new account).setOnClickListener(this);
    }
    public void onItemClick(AdapterView<?> parent, View arg1, int
        position, long arg3) {
        Account account = (Account)parent.getItemAtPosition(position);
        Intent intent = new Intent(this, EmailCpsActivity.class);
      //启动前传值在 Activity 里
       intent.putExtra(EXTRA ACCOUNT , account);
      //启动 Activity
       startActivity(intent);
    }
```

② 定义 onResume()方法，该方法在窗口暂停后回调。所有窗体都继承 Activity 类，因此在窗体设计类中都应该包含此类方法，另外与窗体调用有关的方法有 onStart()方法、onCreate()方法、onPause()方法、onStop()方法、onRestart()方法和 onDestroy()方法。

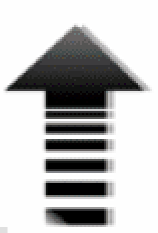
```
    public void onClick(View v) {
        if (v.getId() == R.id.add new account) {
            Intent intent = new Intent(this, AccountSetupActivity.class);
            intent.setFlags(Intent.FLAG ACTIVITY CLEAR TOP);
          startActivity(intent); }
    }
      //暂停后调用
       public void onResume() {
           super.onResume();
           refresh();
       }
      //刷新操作
       private void refresh() {
           Account[] accounts = Preferences.getPreferences(this).getAccounts();
           getListView().setAdapter(new AccountsAdapter(accounts));
       }
@Override
    class AccountsAdapter extends ArrayAdapter<Account> {
       public AccountsAdapter(Account[] accounts) {
           super(WelActivity.this, 0, accounts);
       }
```

```
        public View getView(int position, View convertView, ViewGroup parent) {
            Account account = getItem(position);
            View view;
            if (convertView != null) {
                view = convertView;
            }else {
       view = getLayoutInflater().inflate(R.layout.accounts item, parent, false);
            }
            AccountViewHolder holder = (AccountViewHolder) view.getTag();
            if (holder == null) {
                holder = new AccountViewHolder();
                holder.description = (TextView) view.findViewById
                     (R.id.description);
                holder.email = (TextView) view.findViewById(R.id.email);
                view.setTag(holder);
            }
            holder.description.setText(account.getDescription());
            holder.email.setText(account.getEmail());
            if (account.getEmail().equals(account.getDescription())) {
                holder.email.setVisibility(View.GONE);
            }
            return view;
        }
        class AccountViewHolder {
            public TextView description;
            public TextView email;
        }
    }
}
}
```

(2) Android 是可视化界面开发，每个窗口都有唯一的布局 XML 配置文件，窗口的各种布局效果都有对应的标签表示。比如图像、文字和控件位置的设置等，程序在运行时读取配置文件，满足不同的界面应用。

本实例主界面的布局文件是 AndroidManifest.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
    <ListView
        android:id="@android:id/list"
        android:layout width="fill parent"
        android:layout height="wrap content"
        android:layout weight="1.0"
        />
    <LinearLayout
```



```
        android:id="@+id/empty"
        android:layout_width="fill_parent"
        android:layout height="fill parent"
        android:orientation="vertical">
        <TextView
            android:layout width="fill parent"
            android:layout height="wrap content"
            android:textSize="20sp"
            android:text="@string/accounts welcome"
            android:textColor="?android:attr/textColorPrimary" />
        <View
            android:layout width="fill parent"
            android:layout height="0px"
            android:layout weight="1" />
    </LinearLayout>
    <RelativeLayout
            android:layout width="fill parent"
            android:layout height="54dip"
            android:background="@android:drawable/menu_full_frame">
            <Button
                android:id="@+id/add new account"
                android:layout width="wrap content"
                android:minWidth="100dip"
                android:layout height="wrap content"
                android:text="@string/next action"
                android:drawableRight="@drawable/button indicator next"
                android:layout_alignParentRight="true"
                android:layout centerVertical="true" />
        </RelativeLayout>
</LinearLayout>
```

以上代码中采用 LinearLayout 布局，android:orientation="vertical"实现控件水平方向排列，android:orientation="horizontal"实现控件竖直排列。标签 RelativeLayout 布局提供一个容器，所有控件在容器中的位置按相对位置来计算。

Android:id 定义组件的 ID，程序根据 ID 可以访问相应的的控件。代码中定义了 list、empty 和 add_new_account，分别表示 ListView 组件、LinearLayout 组件和 Button 组件。android:layout_width="fill_parent" 和 android:layout_height="wrap_content"表示控制宽度占全屏，控制的高度适应容器的大小。总之，fill_parent 就是让控件宽或者高占全屏，而 wrap_content 是让控件的高或宽仅仅把控件里的内容包裹住，而不是全屏。

16.4.2　系统主界面

系统根据输入的用户名和密码，设置用户的属性，如图 16-10 所示。

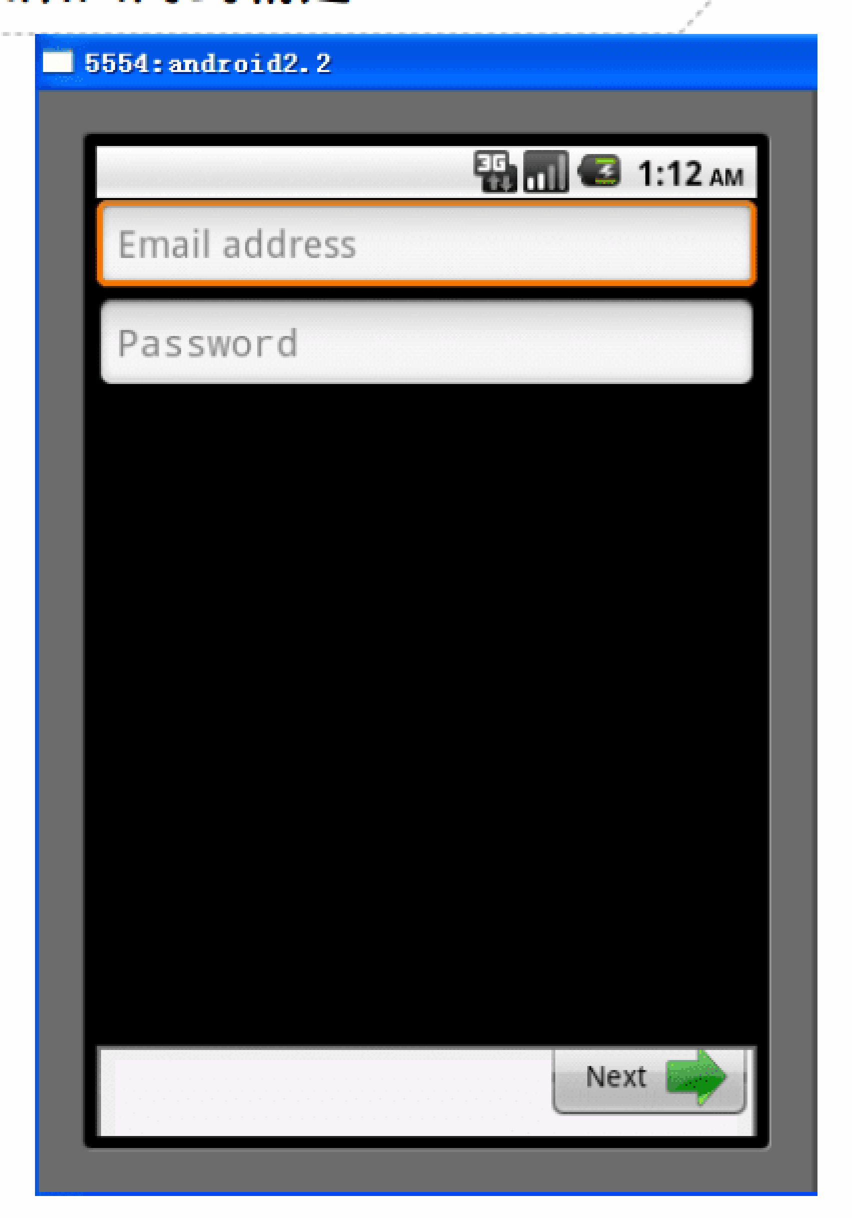


图 16-10　用户设置

(1) 编写文件 AccountSetupActivity.java，在此定义用户设置界面，主要代码如下。

① 定义 onCreate 方法初始化窗体，方法参数 savedInstanceState 保存当前 Activity 的状态信息。定义监听器 setOnClickListener()、addTextChangedListener 和 addTextChangedListener()。对应代码如下：

```
public class AccountSetupActivity extends Activity implements
       OnClickListener,TextWatcher{
    private final static int DIALOG NOTE = 1;
    private EmailAddressValidator mEmailValidator = new
        EmailAddressValidator();
   private EditText mEmailView;
   private EditText mPasswordView;
   private Button mNextButton;
   private Account mAccount;
   private Provider mProvider;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      //加载 activity_account_setup。XML 布局文件
      setContentView(R.layout.activity_account_setup);
      mEmailView = (EditText)findViewById(R.id.account email);
      mPasswordView = (EditText)findViewById(R.id.account password);
      mNextButton = (Button)findViewById(R.id.next);
      //定义监听器
      mNextButton.setOnClickListener(this);
      mEmailView.addTextChangedListener(this);
      mPasswordView.addTextChangedListener(this);
   }
```

② 定义 onCreateDialog()方法创建一个对话框，方法参数表示对话框 ID。



AlertDialog.Builder(this)创建一个 AlertDialog 对话框，setIcon()方法设置对话框图片，setTitle()方法设置显示标题，setMessage()方法定义提示信息内容。setPositiveButton()方法设置确定按钮的一些属性，第一个参数为按钮上的显示内容；第二个参数为 DialogInterface.OnClickListener()监听器对象，监听单击事件。对应代码如下：

```
//创建一个对话框
public Dialog onCreateDialog(int id) {
    if (id == DIALOG NOTE) {
        if (mProvider != null && mProvider.note != null) {
            return new AlertDialog.Builder(this)
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setTitle(android.R.string.dialog alert title)
                .setMessage(mProvider.note)
                .setPositiveButton(
                        getString(R.string.okay action),
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                int which) {
                                finishAutoSetup();
                            }
                        })
                .setNegativeButton(
                        getString(R.string.cancel action),
                        null)
                .create();
        }
    }
    return null;
}
```

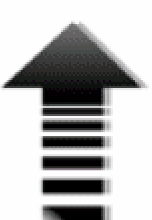
③ 当用户单击“确定”按钮后调用 finishAutoSetup()方法，实例化 URI 对象保存用户邮件的详细信息。如：用户名、密码、主机地址和端口。最终封装在 Account 类的实例 mAccount，调用对应的 set()方法赋值。邮件用户设置不正确时调用 onManualSetup()方法，若用户设置正确时则调用 actionCheckSettings()方法。对应代码如下：

```
private void finishAutoSetup() {
    String email = mEmailView.getText().toString().trim();
    String password = mPasswordView.getText().toString().trim();
    String[] emailParts = email.split("@");
    String user = emailParts[0];
    String domain = emailParts[1];
    URI incomingUri = null;
    URI outgoingUri = null;
    try {
        String incomingUsername = mProvider.incomingUsernameTemplate;
        incomingUsername = incomingUsername.replaceAll("\\$email", email);
        incomingUsername = incomingUsername.replaceAll("\\$user", user);
        incomingUsername = incomingUsername.replaceAll("\\$domain", domain);
        URI incomingUriTemplate = mProvider.incomingUriTemplate;
        incomingUri = new URI(incomingUriTemplate.getScheme(),
```

```
        incomingUsername + ":"
                + password, incomingUriTemplate.getHost(),
                  incomingUriTemplate.getPort(), null,
                  null, null);
        String outgoingUsername = mProvider.outgoingUsernameTemplate;
        outgoingUsername = outgoingUsername.replaceAll("\\$email", email);
        outgoingUsername = outgoingUsername.replaceAll("\\$user", user);
        outgoingUsername = outgoingUsername.replaceAll("\\$domain", domain);
        URI outgoingUriTemplate = mProvider.outgoingUriTemplate;
        outgoingUri = new URI(outgoingUriTemplate.getScheme(),
        outgoingUsername + ":"
                + password, outgoingUriTemplate.getHost(),
                  outgoingUriTemplate.getPort(), null,
                  null, null);
    } catch (URISyntaxException use) {
        onManualSetup();
        return;
    }
  //给 Account 对象的属性赋值
    mAccount = new Account(this);
    mAccount.setName(getOwnerName());
    mAccount.setEmail(email);
    mAccount.setStoreUri(incomingUri.toString());
    mAccount.setSenderUri(outgoingUri.toString());
  mAccount.setDraftsFolderName(getString(R.string.special mailbox name drafts));
  mAccount.setTrashFolderName(getString(R.string.special mailbox name trash));
  mAccount.setOutboxFolderName(getString(R.string.special mailbox name outbox));
    mAccount.setSentFolderName(getString(R.string.special mailbox name sent));
    if (incomingUri.toString().startsWith("imap")) {
        mAccount.setDeletePolicy(Account.DELETE POLICY ON DELETE);
    }
    AccountCheckSettings.actionCheckSettings(this, mAccount, true, true);
  }
```

④ 定义 getOwnerName()方法获取当前用户，通过共享数据接口取得用户 Account 对象。getName()返回具体的用户名。单击“向下”按钮调用 findProviderForDomain()方法，从 providers_product.xml 配置文件中读取已有账户信息。对应代码如下：

```
  private String getOwnerName() {
    String name = null;
    //通过 SharedPreferences 对象取得用户名
    Account account = Preferences.getPreferences(this).getDefaultAccount();
    if (account != null) {
        name = account.getName();
    }
    return name;
  }
  private void onNext() {
    String email = mEmailView.getText().toString().trim();
    String[] emailParts = email.split("@");
    String domain = emailParts[1].trim();
```



```
        mProvider = findProviderForDomain(domain);
        if (mProvider == null) {
             //默认设置用户调用 manual
            onManualSetup();
            return;
        }
        if (mProvider.note != null) {
             //显示对话框
            showDialog(DIALOG_NOTE);
        }
        else {
            finishAutoSetup();
        }
    }
    private Provider findProviderForDomain(String domain) {
        Provider p = findProviderForDomain(domain,
             R.xml.providers_product);
        if (p == null) {
            p = findProviderForDomain(domain, R.xml.providers);
        }
        return p;
    }
```

⑤ 如果 providers_product 文件中没有用户信息，通过 findProviderForDomain()方法读取 Providers 文件中提供接收邮件和发送邮件的服务器的信息。最后将 id、lable、domain、uri 保存在 Provider 实例中。对应代码如下：

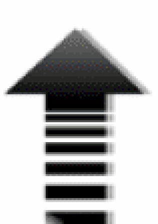
```
   private String getXmlAttribute(XmlResourceParser xml, String name) {
        int resId = xml.getAttributeResourceValue(null, name, 0);
        if (resId == 0) {
            return xml.getAttributeValue(null, name);
        }
        else {
            return getString(resId);
        }
    }
    //读取资源文件 XML
    private Provider findProviderForDomain(String domain, int resourceId)
{
        try {
            XmlResourceParser xml = getResources().getXml(resourceId);
            int xmlEventType;
            Provider provider = null;
            //逐行读取 XML 文件
            while ((xmlEventType = xml.next()) != XmlResourceParser.END_DOCUMENT) {
                if (xmlEventType == XmlResourceParser.START_TAG
                        && "provider".equals(xml.getName())
                        && domain.equalsIgnoreCase(getXmlAttribute(xml,
                            "domain"))) {
                    provider = new Provider();
                    //读取指定键值的值
```




```
                provider.id = getXmlAttribute(xml, "id");
                provider.label = getXmlAttribute(xml, "label");
                provider.domain = getXmlAttribute(xml, "domain");
                provider.note = getXmlAttribute(xml, "note");
            }
            else if (xmlEventType == XmlResourceParser.START_TAG
                    && "incoming".equals(xml.getName())
                    && provider != null) {
            provider.incomingUriTemplate = new URI(getXmlAttribute(xml, "uri"));
            provider.incomingUsernameTemplate = getXmlAttribute(xml, "username");
            }
            else if (xmlEventType == XmlResourceParser.START_TAG
                    && "outgoing".equals(xml.getName())
                    && provider != null) {
            provider.outgoingUriTemplate = new URI(getXmlAttribute(xml, "uri"));
            provider.outgoingUsernameTemplate = getXmlAttribute(xml, "username");
            }
            else if (xmlEventType == XmlResourceParser.END_TAG
                    && "provider".equals(xml.getName())
                    && provider != null) {
                return provider;
            }
        }
    }
    catch (Exception e) {
        Log.e(Email.LOG_TAG, "Error while trying to load provider settings.", e);
    }
    return null;
}
```

⑥ 定义 onManualSetup()方法重新设置用户名和密码，将新设定的信息封装在 URI 实例中。若设定失败，makeText()方法在主界面显示出错内容。设定用户名和密码后进入 AccountSetupAccountType 对象的 actionSelectAccountType()方法。对应代码如下：

```
private void onManualSetup() {
    String email = mEmailView.getText().toString().trim();
    String password = mPasswordView.getText().toString().trim();
    String[] emailParts = email.split("@");
    String user = emailParts[0].trim();
    String domain = emailParts[1].trim();
    mAccount = new Account(this);
    mAccount.setName(getOwnerName());
    mAccount.setEmail(email);
    try { //实例化 URL 实例，为 URL 赋值
        URI uri = new URI("placeholder", user + ":" + password, domain,
            -1, null, null, null);
        mAccount.setStoreUri(uri.toString());
        mAccount.setSenderUri(uri.toString());
    } catch (URISyntaxException use) {
      //URL 地址出错将提示
      Toast.makeText(this,R.string.account_setup_username_password_
```



```
            toast, Toast.LENGTH LONG).show();
            mAccount = null;
            return;
        } //为 Account 对象的属性赋值
    mAccount.setDraftsFolderName(getString(R.string.special_mailbox_name_drafts));
    mAccount.setTrashFolderName(getString(R.string.special mailbox name trash));
    mAccount.setOutboxFolderName(getString(R.string.special mailbox name outbox));
        mAccount.setSentFolderName(getString(R.string.special mailbox name sent));
        AccountSetupAccountType.actionSelectAccountType(this, mAccount, true);
        finish();
    }
```

⑦ 定义 onActivityResult()方法接收处理结果，当执行完 finish()后，Activity 执行结束，并且将返回值返回给调用它的父类 Activity 类。onActivityResult()方法的第一个参数表示 Activity 请求码，第二个参数表示返回结果，结果码最常用的有 RESULT_OK 和 RESULT_CANCELED，前者表示执行成功，后者表示取消操作。对应代码如下：

```
   //根据指定返回码执行 Activity
    protected void onActivityResult(int requestCode, int resultCode,
            android.content.Intent data) {
        if (resultCode == RESULT OK) {
            mAccount.setDescription(mAccount.getEmail());
            mAccount.save(Preferences.getPreferences(this));
            Preferences.getPreferences(this).setDefaultAccount(mAccount);
            AccountSetupNames.actionSetNames(this, mAccount);
            finish();
        }
    }
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE NO TITLE);
        setContentView(R.layout.main);
        systemProvider=new SystemService(this);
        cursor=systemProvider.allSongs();
        //读 MUSIC 键值的值
        SharedPreferences sp = getSharedPreferences("MUSIC",MODE WORLD READABLE);
        if (sp != null) {
            playingName = sp.getString("PLAYINGNAME", null);
            selectName = sp.getString("SELECTNAME", null);
            String s = sp.getString("MUSIC LIST", null);
            if (s != null)
                music List = StringHelper.spiltString(s);
        }
```

(2) 系统主界面的布局文件是 activity_account_setup.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout width="match parent"
  android:layout_height="match_parent"
```

```
  android:orientation="vertical">
  <EditText
       android:id="@+id/account_email"
       android:hint="@string/account_setup_basics_email_hint"
       android:inputType="textEmailAddress"
       android:imeOptions="actionNext"
       android:layout_height="wrap_content"
       android:layout_width="fill_parent"
       />
  <EditText
       android:id="@+id/account_password"
       android:hint="@string/account_setup_basics_password_hint"
       android:inputType="textPassword"
       android:imeOptions="actionDone"
       android:layout_height="wrap_content"
       android:layout_width="fill_parent"
       android:nextFocusDown="@+id/next"
   />
   <View
       android:layout_width="fill_parent"
       android:layout_height="0px"
       android:layout_weight="1"
       />
   <RelativeLayout
       android:layout_width="fill_parent"
       android:layout_height="54dip"
       android:background="@android:drawable/menu_full_frame"
       >
       <Button
           android:id="@+id/next"
           android:text="@string/next_action"
           android:layout_height="wrap_content"
           android:layout_width="wrap_content"
           android:minWidth="100dip"
           android:drawableRight="@drawable/button_indicator_next"
           android:layout_alignParentRight="true"
           android:layout_centerVertical="true"
           />
    </RelativeLayout>
</LinearLayout>
```

以上代码 layout_height="match_parent"中，match_parent 和 fill_parent 其实效果一样。

nextFocusDown 定义单击 Down 键时，account_password 文本框获得焦点。nextFocusUp 定义单击 Up 键时某组件获得焦点、nextFocusLeft 定义单击 Left 键时某组件获得焦点和 nextFocusRight 定义点击 Right 键时某组件获得焦点。

inputType 定义该组件是输入框类型。

imeOptions 指定输入法窗口中的回车键功能，actionDone 表示软键盘下方变成“完成”，单击后光标保持在原来的输入框上，并且软键盘关闭。其他可选值为 normal、actionNext、actionSearch 等。

16.4.3 邮箱类型设置

在输入用户名和密码后，单击 Next 按钮，将弹出邮箱类型设置界面，如图 16-11 所示。

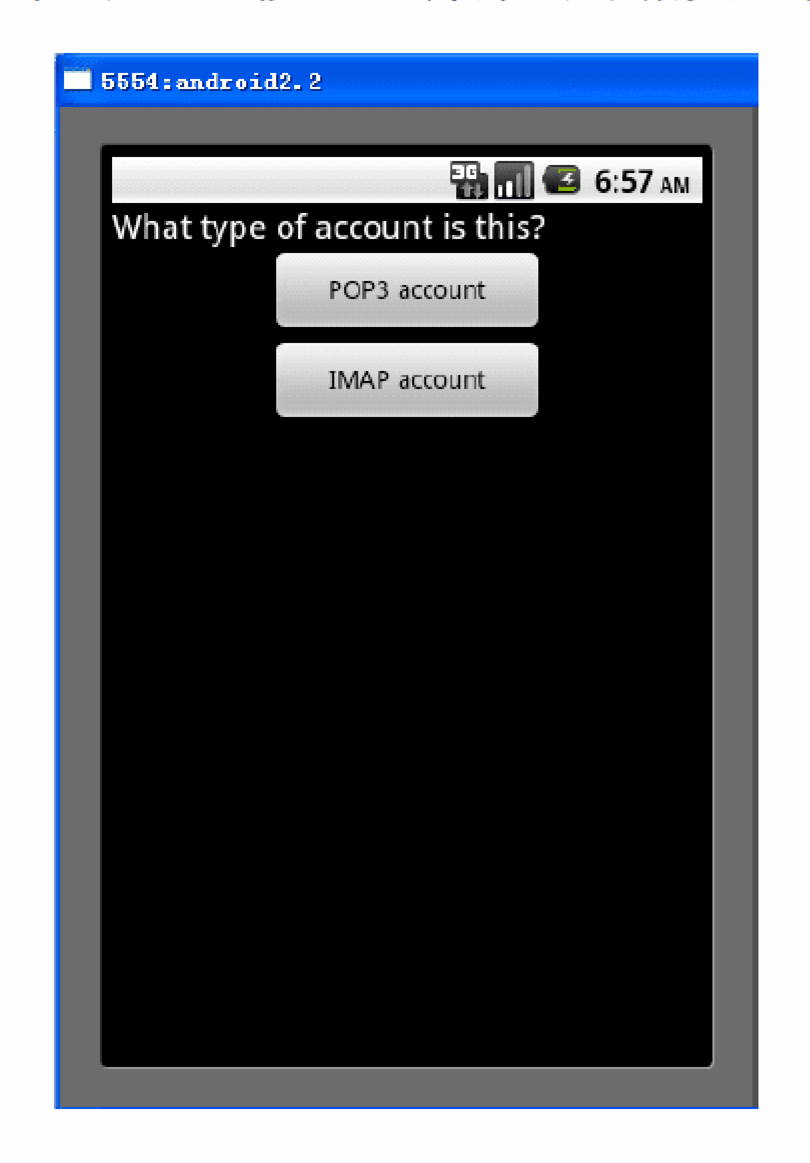


图 16-11 邮箱类型设置界面

(1) 编写文件 AccountSetupAccountType.java，在此定义邮箱类型设置界面，主要代码如下。

① 定义 onCreate 方法初始化窗体，为 Button 对象定义监听器 setOnClickListener()。其中 Context 参数将接收从主界面窗体传送的数据，利用 actionSelectAccountType()方法做初始化操作，Intent()方法使程序执行跳转到 AccountSetupAccountType 实例。putExtra()方法以键值对的形式保存数据。对应代码如下：

```
public class AccountSetupAccountType extends Activity implements
OnClickListener {
   private static final String EXTRA_ACCOUNT = "account";
   private static final String EXTRA_MAKE_DEFAULT = "makeDefault";
   private Account mAccount;
   private boolean mMakeDefault;
   //初始化
  public static void actionSelectAccountType(Context context, Account
        account, boolean makeDefault) {
        Intent i = new Intent(context, AccountSetupAccountType.class);
        //为 EXTRA_ACCOUNT 指定的键赋值
        i.putExtra(EXTRA_ACCOUNT, account);
        i.putExtra(EXTRA_MAKE_DEFAULT, makeDefault);
        //启动 Activity
        context.startActivity(i);
   }
   //创建一个窗体
   public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        //加载 activity_account_setup_type 布局 XML 文件
        setContentView(R.layout.activity account setup type);
        ((Button)findViewById(R.id.pop)).setOnClickListener(this);
        ((Button)findViewById(R.id.imap)).setOnClickListener(this);
        mAccount = (Account)getIntent().getSerializableExtra(EXTRA ACCOUNT);
        mMakeDefault = (boolean)getIntent().getBooleanExtra
            (EXTRA MAKE DEFAULT, false);
    }
```

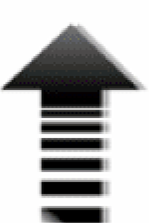
② 定义 onPop()方法保存用户的 URL 地址，getUserInfo()方法取得用户，getHost()方法取得主机地址，getPort()方法取得端口。此处的 uri 实例对象表示用户类型是 pop3 协议(允许用户从服务器上把邮件存储到本地主机)。对应代码如下所示。

```
    @Override
    private void onPop() {
        try {
             //定义并为 URI 实例赋值
            URI uri = new URI(mAccount.getStoreUri());
            uri = new URI("pop3", uri.getUserInfo(), uri.getHost(),
                uri.getPort(), null, null, null);
            mAccount.setStoreUri(uri.toString());
        } catch (URISyntaxException use) {
            throw new Error(use);
        }
        AccountSetupIncoming.actionIncomingSettings(this, mAccount,
            mMakeDefault);
        //执行 Activity
        finish();

    }
```

③ 定义 onImap()方法保存用户的 URL 地址，此处的 URI 实例对象表示用户类型是 imap 协议(允许用户在线与邮件服务器交互信息)。无论是采用哪种通信协议都要调用 actionIncomingSettings()方法进入用户收取邮件界面设置，onClick()监听用户单击按钮的动作。对应代码如下：

```
    private void onImap() {
        try {
            //定义并为 URI 实例赋值
             URI uri = new URI(mAccount.getStoreUri());
                 uri = new URI("imap", uri.getUserInfo(), uri.getHost(),
                 uri.getPort(), null, null, null);
            mAccount.setStoreUri(uri.toString());
        } catch (URISyntaxException use) {
            throw new Error(use);
        }
        mAccount.setDeletePolicy(Account.DELETE POLICY ON DELETE);
        AccountSetupIncoming.actionIncomingSettings(this, mAccount, mMakeDefault);
     //执行 Activity
        finish();
```



```
    }
  //根据触发的组件调用相应的方法
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.pop:
                onPop();
                break;
            case R.id.imap:
                onImap();
                break;
        }
    }
}
```

(2) 邮箱类型设置的布局文件是 activity_account_setup_type.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="fill parent"
      android:layout height="fill parent"
    android:orientation="vertical"
      >
      <TextView
          android:text="@string/account setup account type instructions"
        android:layout height="wrap content"
        android:layout_width="fill_parent"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="?android:attr/textColorPrimary"
          />
    <Button
        android:id="@+id/pop"
        android:text="@string/account setup account type pop action"
        android:layout height="wrap content"
        android:layout width="150dip"
        android:minWidth="100dip"
        android:layout gravity="center horizontal"
        />
    <Button
        android:id="@+id/imap"
        android:text="@string/account setup account type imap action"
        android:layout height="wrap content"
        android:layout width="150sp"
        android:minWidth="100dip"
        android:layout gravity="center horizontal"
        />
</LinearLayout>
```

以上代码中“android:textAppearance="?android:attr/textAppearanceMedium"”引用的是系统自带的一个外观，“？”表示系统是否有这种外观，否则使用默认的外观。Android 系统自带的文字外观设置及实际显示效果图的设置有 textappearancebutton、textappearanceinverse、textappearancelarge、textappearancelargeinverse、textappearancemedium、

extappearancesmallinverse、textappearancemediuminverse 和 textappearancesmall。

“android:textColor="?android:attr/textColorPrimary"”同样引用的是系统自带的一个外观。设置界面背景及文字颜色最常用的两种方法：直接在布局文件中设置如“android:backgound="#FFFFFFFF", android:textcolor="#00000000"”和把颜色提取出来形成资源，放在资源文件(如 values/drawable/color.xml)下面。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<drawable name="white">#FFFFFFFF</drawable>
<drawable name="black">#FF000000</drawable>
</resources>
```

然后在布局文件中使用代码 android:backgound="@drawable/white", android:textcolor= "@drawable/black" 或者在 Java 文件中通过代码 setBackgroundColor(int color)，setBackgroundResource(int resid)、setTextColor(int color)来设置。

16.4.4　邮箱收取设置

在确定邮件类型后，单击 POP3 Account 或者 Imap Account 按钮，将弹出邮箱收取设置界面，如图 16-12 所示。

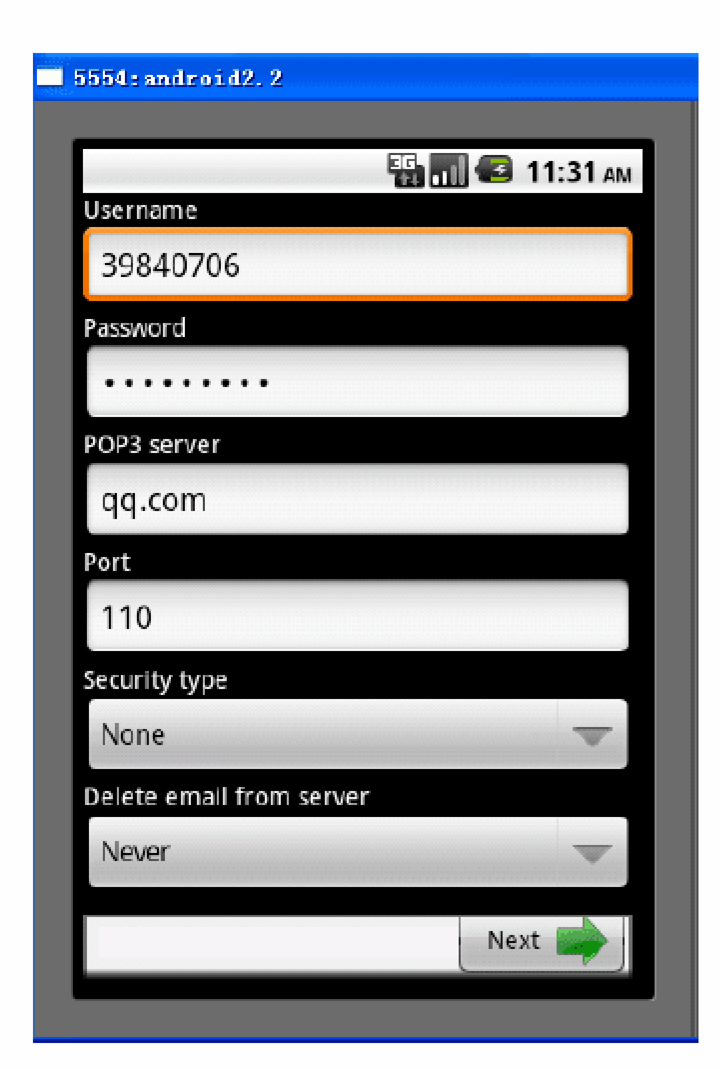
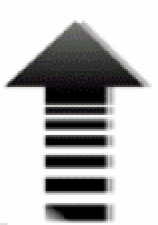


图 16-12　邮箱收取设置界面

(1) 此邮箱收取设置界面功能是通过文件 AccountSetupIncoming.java 实现的，接下来开始讲解此文件的实现流程。

① 定义 actionIncomingSettings()方法和 actionEditIncomingSettings()方法做数据初始化操作，其中 Context 参数将接收从主界面窗体传送的数据，Intent()方法使程序执行跳转到 AccountSetupIncoming 实例，putExtra()方法以键值对的形式保存数据。startActivity()方法启动在不同 Activity 间切换。对应代码如下：

```
private static final String EXTRA_ACCOUNT = "account";
 private static final String EXTRA_MAKE_DEFAULT = "makeDefault";
```



```
 //初始化端口选项
  private static final int popPorts[] = {
         110, 995, 995, 110, 110
  };
  private static final String popSchemes[] = {
         "pop3", "pop3+ssl", "pop3+ssl+", "pop3+tls", "pop3+tls+"
  };
  private static final int imapPorts[] = {
         143, 993, 993, 143, 143
  };
  private static final String imapSchemes[] = {
         "imap", "imap+ssl", "imap+ssl+", "imap+tls", "imap+tls+"
  };
  private int mAccountPorts[];
  private String mAccountSchemes[];
  private EditText mUsernameView;
  private EditText mPasswordView;
  private EditText mServerView;
  private EditText mPortView;
  private Spinner mSecurityTypeView;
  private Spinner mDeletePolicyView;
  private EditText mImapPathPrefixView;
  private Button mNextButton;
  private Account mAccount;
  private boolean mMakeDefault;
  public static void actionIncomingSettings(Activity context, Account
     account, boolean makeDefault) {
     // 定义 Intent 对象，供 Activity 跳转
     Intent i = new Intent(context, AccountSetupIncoming.class);
     i.putExtra(EXTRA ACCOUNT, account);
     i.putExtra(EXTRA MAKE DEFAULT, makeDefault);
   //启动 Activity(
     context.startActivity(i);
  }
public static void actionEditIncomingSettings(Activity context, Account
     account) {
     Intent i = new Intent(context, AccountSetupIncoming.class);
     i.setAction(Intent.ACTION EDIT);
     i.putExtra(EXTRA ACCOUNT, account);
     context.startActivity(i);
  }
```

② Android 开发中的四大组件包含：活动(Activity)、服务(Services)、广播接收者(BroadcastReceiver)和内容提供者(ContentProvider)，活动(Activity)是一个很重要的部分，表示一个可视化的用户界面，关注用户从事的事件，几乎所有的活动都是要和用户进行交互。

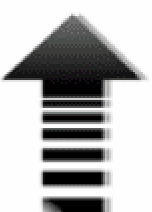
③ 定义 onCreate 方法初始化窗体，定义了 Spinner 控件，该控件主要就是一个列表。Spinner 是 View 类的一个子类，利用数组的赋值方式为其写入初值。对应代码如下。

```
  @Override
  public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
       //加载 activity_account_setup_incoming 布局 XML 文件
        setContentView(R.layout.activity account setup incoming);
        mUsernameView = (EditText)findViewById(R.id.account username);
        mPasswordView = (EditText)findViewById(R.id.account_password);
        TextView serverLabelView = (TextView) findViewById
             (R.id.account server label);
        mServerView = (EditText)findViewById(R.id.account server);
        mPortView = (EditText)findViewById(R.id.account port);
        mSecurityTypeView = (Spinner)findViewById(R.id.account security type);
        mDeletePolicyView = (Spinner)findViewById(R.id.account delete policy);
        mImapPathPrefixView = (EditText)findViewById(R.id.imap path prefix);
        mNextButton = (Button)findViewById(R.id.next);
       //绑定监听器
        mNextButton.setOnClickListener(this);
        SpinnerOption securityTypes[] = {
               new SpinnerOption(0, getString(R.string.account setup
                   incoming security none label)),
               new SpinnerOption(1, getString(R.string.account setup
                   incoming security ssl optional label)),
               new SpinnerOption(2, getString(R.string.account setup
                   incoming security ssl label)),
               new SpinnerOption(3, getString(R.string.account setup
                   incoming security tls optional label)),
               new SpinnerOption(4, getString(R.string.account setup
                   incoming security tls label)),
        };
        SpinnerOption deletePolicies[] = {
               new SpinnerOption(0,
            getString(R.string.account setup incoming delete policy never label)),
               new SpinnerOption(1,
            getString(R.string.account setup incoming delete policy 7days label)),
               new SpinnerOption(2,
            getString(R.string.account setup incoming delete policy delete label)),
        };
```

④ ArrayAdapter 是从 BaseAdapter 派生出来的，具备 BaseAdapter 的所有功能，但 ArrayAdapter 更为强大，它在实例化时可以直接使用泛型构造。ArrayAdapter 分三种显示模式，分别是简单的、样式丰富的但内容简单的和内容丰富的。Android SDK 中可以看到 android.widget.ArrayAdapter<T>形式，ArrayAdapter(Context context, int textViewResourceId) 第二个参数直接绑定一个 layout。对应代码如下：

```
ArrayAdapter<SpinnerOption> securityTypesAdapter = new
ArrayAdapter<SpinnerOption>(this,
             android.R.layout.simple spinner item, securityTypes);
             securityTypesAdapter.setDropDownViewResource
             (android.R.layout.simple spinner dropdown item);
              mSecurityTypeView.setAdapter(securityTypesAdapter);
             ArrayAdapter<SpinnerOption> deletePoliciesAdapter = new
             ArrayAdapter<SpinnerOption>(this,
```



```
                android.R.layout.simple spinner item, deletePolicies);
            deletePoliciesAdapter
               .setDropDownViewResource(android.R.layout.simple spinner
                      dropdown item);
            mDeletePolicyView.setAdapter(deletePoliciesAdapter);
            mSecurityTypeView.setOnItemSelectedListener(new
                 AdapterView.OnItemSelectedListener() {
            public void onItemSelected(AdapterView arg0, View arg1, int
                arg2, long arg3) {
                updatePortFromSecurityType();
        }
        public void onNothingSelected(AdapterView<?> arg0) {
        }
    });
```

⑤ TextWatcher 实例监控 EditText 组件输入的内容发生的变化，然后定义 addTextChangedListener 监听器分别监控用户名、密码、服务和端口是否有输入内容，若没全部输入内容，则 Next 按钮将成不可用状态。对应代码如下：

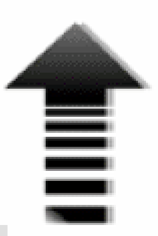
```
    TextWatcher validationTextWatcher = new TextWatcher() {
        public void afterTextChanged(Editable s) {
            validateFields();
        }
    };
   //定义监听器
    mUsernameView.addTextChangedListener(validationTextWatcher);
    mPasswordView.addTextChangedListener(validationTextWatcher);
    mServerView.addTextChangedListener(validationTextWatcher);
    mPortView.addTextChangedListener(validationTextWatcher);
    if (savedInstanceState != null && savedInstanceState.containsKey
        (EXTRA ACCOUNT)) {
  //取得 mAccount 实例
        mAccount = (Account)savedInstanceState.getSerializable(EXTRA ACCOUNT);
    }
    try {
        URI uri = new URI(mAccount.getStoreUri());
        String username = null;
        String password = null;
        if (uri.getUserInfo() != null) {
            String[] userInfoParts = uri.getUserInfo().split(":", 2);
            username = userInfoParts[0];
            if (userInfoParts.length > 1) {
                password = userInfoParts[1];
            }
        }
        if (username != null) {
            mUsernameView.setText(username);
        }
        if (password != null) {
            mPasswordView.setText(password);
        }
```

⑥ getScheme()方法返回当前请求所使用的协议。根据返回结果为 mAccountPorts 变量设置相应的值。对应代码如下：

```
        if
            (uri.getScheme().startsWith("pop3")) { serverLabelView.setText
            (R.string.account setup incoming pop server label);
              mAccountPorts = popPorts;
              mAccountSchemes = popSchemes;
              findViewById(R.id.imap path prefix section).setVisibility
              (View.GONE);
           } else if (uri.getScheme().startsWith("imap"))
              { serverLabelView.setText(R.string.account setup
              incoming imap server label);
              mAccountPorts = imapPorts;
              mAccountSchemes = imapSchemes; findViewById
               (R.id.account delete policy label).setVisibility
               (View.GONE);
              mDeletePolicyView.setVisibility(View.GONE);
              if (uri.getPath() != null && uri.getPath().length() > 0) {
                 mImapPathPrefixView.setText(uri.getPath().substring(1));
              }
           } else {
              throw new Error("Unknown account type: " + mAccount.getStoreUri());
           }
           for (int i = 0; i < mAccountSchemes.length; i++) {
              if (mAccountSchemes[i].equals(uri.getScheme())) {
                 SpinnerOption.setSpinnerOptionValue(mSecurityTypeView, i);
              }
           }
           SpinnerOption.setSpinnerOptionValue(mDeletePolicyView,
               mAccount.getDeletePolicy());
           if (uri.getHost() != null) {
              mServerView.setText(uri.getHost());
           }
           if (uri.getPort() != -1) {
              mPortView.setText(Integer.toString(uri.getPort()));
           } else {
              updatePortFromSecurityType();
           }
        } catch (URISyntaxException use) {
           throw new Error(use);
        }
      //检查组件里内容的变化
        validateFields();
    }
```

(2) 邮箱收取界面的布局文件是 activity_account_setup_incoming.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
   xmlns:android="http://schemas.android.com/apk/res/android"
```



```
    android:layout width="fill parent"
    android:layout height="fill parent"
    android:scrollbarStyle="outsideInset">
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout height="fill parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/account server label"
            android:text="@string/account setup incoming pop server label"
            android:layout height="wrap content"
            android:layout width="fill parent"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="?android:attr/textColorPrimary" />
        <Spinner
            android:id="@+id/account security type"
            android:layout height="wrap content"
            android:layout width="fill parent" />
        <TextView
            android:id="@+id/account delete policy label"
            android:text="@string/account setup incoming delete policy label"
            android:layout height="wrap content"
            android:layout width="fill parent"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="?android:attr/textColorPrimary" />
        <Spinner
            android:id="@+id/account delete policy"
            android:layout height="wrap content"
            android:layout width="fill parent" />
        <View
            android:layout width="fill parent"
            android:layout height="0px"
            android:layout weight="1" />
        <RelativeLayout
            android:layout width="fill parent"
            android:layout height="54dip"
            android:background="@android:drawable/menu full frame">
            <Button
                android:id="@+id/next"
                android:text="@string/next action"
                android:layout height="wrap content"
                android:layout width="wrap content"
                android:minWidth="100dip"
                android:drawableRight="@drawable/button indicator next"
                android:layout alignParentRight="true"
                android:layout centerVertical="true" />
        </RelativeLayout>
    </LinearLayout>
</ScrollView>
```

以上代码“android:drawableRight="@drawable/button_indicator_next"”中，drawable 在 Android SDK 中的主要作用是：在 XML 中定义各种动画，然后把 XML 当作 drawable 资源来读取，通过 drawable 显示动画。

drawable 就是一个可画的对象，可能是一张位图(bitmap drawable)，也可能是一个图形(shape drawable)，还有可能是一个图层(layer drawable)。开发程序时为了兼容不同平台不同屏幕，所以要求建立多个文件夹根据需求存放不同屏幕版本图片。

在Android SDK 2.1 版本中有drawable-mdpi、drawable-ldpi和drawable-hdpi三个文件夹，这三个文件夹主要是为了支持多分辨率。系统运行时会根据机器的分辨率来分别到这几个文件夹中去找对应的图片。xhdpi是从 Android 2.2 (API Level 8)开始增加的分类。xlarge是从Android 2.3 (API Level 9)增加的分类。在本项目中res/drawable-xhdpi/资源下存放button_indicator_next.png图片，另外在以上三个目录中同样有此名字的图片。

16.4.5 邮箱发送设置

在设置好发送邮件的必要信息后，单击 Next 按钮，将弹出邮箱发送设置界面，如图 16-13。

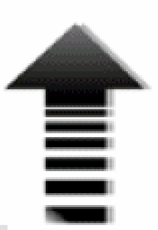


图 16-13 邮箱发送设置界面

(1) 编写文件 AccountSetupOutgoing.java，在此定义邮箱发送设置界面。

① 定义 actionOutgoingSettings()方法和 actionEditOutgoingSettings()方法做数据初始化操作，其中 Context 参数将接收从主界面窗体传送的数据，Intent()方法使程序执行跳转到 AccountSetupOutgoing 实例，putExtra()方法以键值对的形式保存数据。startActivity()方法启动在不同 Activity 之间切换。对应代码如下：

```
public class AccountSetupOutgoing extends Activity implements
OnClickListener,
        OnCheckedChangeListener {
    private static final String EXTRA_ACCOUNT = "account";
```



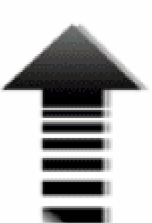
```
    private static final String EXTRA MAKE DEFAULT = "makeDefault";
   //定义发送端口
    private static final int smtpPorts[] = {
           25, 465, 465, 25, 25
    };
    private static final String smtpSchemes[] = {
           "smtp", "smtp+ssl", "smtp+ssl+", "smtp+tls", "smtp+tls+"
    };
    private EditText mUsernameView;
    private EditText mPasswordView;
    private EditText mServerView;
    private EditText mPortView;
    private CheckBox mRequireLoginView;
    private ViewGroup mRequireLoginSettingsView;
    private Spinner mSecurityTypeView;
    private Button mNextButton;
    private Account mAccount;
    private boolean mMakeDefault;
public static void actionOutgoingSettings(Context context, Account
       account, boolean makeDefault) {
  //定义 Intent 实例，将 Activity 设定跳转到 AccountSetupOutgoing
       Intent i = new Intent (context, AccountSetupOutgoing.class);
       i.putExtra(EXTRA ACCOUNT, account);
       i.putExtra(EXTRA MAKE DEFAULT, makeDefault);
       //启动 Activity
       context.startActivity(i);
    }
    public static void actionEditOutgoingSettings(Context context,
       Account account) {
       Intent i = new Intent(context, AccountSetupOutgoing.class);
       i.putExtra(EXTRA ACCOUNT, account);
       context.startActivity(i);
    }
```

② 定义 onCreate(Bundle savedInstanceState)方法初始化窗体，在创建 Activity 时调用。还以 Bundle 的形式提供对以前储存的任何状态的访问。对应代码如下：

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       //加载界面布局文件 activity_account_setup_outgoing.XML
       setContentView(R.layout.activity account setup outgoing);
       mUsernameView = (EditText)findViewById(R.id.account username);
       mPasswordView = (EditText)findViewById(R.id.account password);
       mServerView = (EditText)findViewById(R.id.account server);
       mPortView = (EditText)findViewById(R.id.account port);
       mRequireLoginView = (CheckBox)findViewById(R.id.account require login);
       mRequireLoginSettingsView = (ViewGroup)findViewById
           (R.id.account require login settings);
       mSecurityTypeView = (Spinner)findViewById(R.id.account security type);
       mNextButton = (Button)findViewById(R.id.next);
```




```
mNextButton.setOnClickListener(this);
//定义监听器
mRequireLoginView.setOnCheckedChangeListener(this);
SpinnerOption securityTypes[] = {
        new SpinnerOption(0, getString(R.string.account_setup_
            incoming_security_none_label)),
        new SpinnerOption(1, getString(R.string.account_setup_
            incoming_security_ssl_optional_label)),
        new SpinnerOption(2, getString(R.string.account_setup_
            incoming_security_ssl_label)),
        new SpinnerOption(3,
              getString(R.string.account_setup_incoming_
                  security_tls_optional_label)),
        new SpinnerOption(4, getString(R.string.account_setup_
            incoming_security_tls_label)),
};
ArrayAdapter<SpinnerOption> securityTypesAdapter = new
      ArrayAdapter<SpinnerOption>(this,
        android.R.layout.simple_spinner_item, securityTypes);
        securityTypesAdapter.setDropDownViewResource
        (android.R.layout.simple_spinner_dropdown_item);
mSecurityTypeView.setAdapter(securityTypesAdapter);
mSecurityTypeView.setOnItemSelectedListener(new
        AdapterView.OnItemSelectedListener() {
//树型组件点击后触发的事件
    public void onItemSelected(AdapterView arg0, View arg1, int
        arg2, long arg3) {
        updatePortFromSecurityType();
    }
    public void onNothingSelected(AdapterView<?> arg0) {
    }
});
TextWatcher validationTextWatcher = new TextWatcher() {
    public void afterTextChanged(Editable s) {
        validateFields();
    }
};
//定义监听器
mUsernameView.addTextChangedListener(validationTextWatcher);
mPasswordView.addTextChangedListener(validationTextWatcher);
mServerView.addTextChangedListener(validationTextWatcher);
mPortView.addTextChangedListener(validationTextWatcher);
mPortView.setKeyListener(DigitsKeyListener.getInstance
     ("0123456789"));
mAccount = (Account)getIntent().getSerializableExtra(EXTRA_ACCOUNT);
mMakeDefault = (boolean)getIntent().getBooleanExtra
     (EXTRA_MAKE_DEFAULT, false);
if (savedInstanceState != null && savedInstanceState.containsKey
    (EXTRA_ACCOUNT)) {
    mAccount = (Account)savedInstanceState.getSerializable(EXTRA_ACCOUNT);
}
```



```
        validateFields();
    }
private void validateFields() {
        boolean enabled =
            Utility.requiredFieldValid(mServerView) && Utility.requiredFieldValid
                 (mPortView);
        if (enabled && mRequireLoginView.isChecked()) {
            enabled = (Utility.requiredFieldValid(mUsernameView)
                    && Utility.requiredFieldValid(mPasswordView));
        }
        if (enabled) {
            try {
                URI uri = getUri();
            } catch (URISyntaxException use) {
                enabled = false;
            }
        }
        mNextButton.setEnabled(enabled);
        Utility.setCompoundDrawablesAlpha(mNextButton, enabled ? 255 : 128);
    }
```

③ 定义 TextWatcher 实例监控 EditText 组件输入的内容发生的变化，然后定义 addTextChangedListener 监听器分别监控用户名、密码、服务和端口是否有输入内容，若没全部输入内容，则 Next 按钮将成不可用状态。对应代码如下：

```
  TextWatcher validationTextWatcher = new TextWatcher() {
            public void afterTextChanged(Editable s) {
                validateFields();
            }
    public void beforeTextChanged(CharSequence s, int start, int count,
        int after) {
            }
      public void onTextChanged(CharSequence s, int start, int before,
         int count) {
            }
        };
        mUsernameView.addTextChangedListener(validationTextWatcher);
        mPasswordView.addTextChangedListener(validationTextWatcher);
        mServerView.addTextChangedListener(validationTextWatcher);
        mPortView.addTextChangedListener(validationTextWatcher);
```

④ 定义 updatePortFromSecurityType()方法，将用户输入的端口号赋值为 mPortView 变量。代码 mSecurityTypeView.getSelectedItem()).value 读取 View 的节点值，onActivityResult()方法检查若 Activity 执行是否成功。对应代码如下：

```
private void updatePortFromSecurityType() {
        int securityType = (Integer)((SpinnerOption)mSecurityTypeView.
            getSelectedItem()).value;
        mPortView.setText(Integer.toString(smtpPorts[securityType]));
    }
    @Override
```

```
// onActivityResult 执行完 Activity 后，将结果返回给调用它的父 Activity
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode == RESULT_OK) {
            if (Intent.ACTION_EDIT.equals(getIntent().getAction())) {
                mAccount.save(Preferences.getPreferences(this));
                finish();
            } else {
                AccountSetupOptions.actionOptions(this, mAccount, mMakeDefault);
                finish();
            }
        }
    }
    private URI getUri() throws URISyntaxException {
        int securityType = (Integer)((SpinnerOption)mSecurityTypeView.
            getSelectedItem()).value;
        String userInfo = null;
        if (mRequireLoginView.isChecked()) {
            userInfo = mUsernameView.getText().toString().trim() + ":"
                    + mPasswordView.getText().toString().trim();
        }
        URI uri = new URI(
                smtpSchemes[securityType],
                userInfo,
                mServerView.getText().toString().trim(),
                Integer.parseInt(mPortView.getText().toString().trim()),
                null, null, null);
        return uri;
    }
```

⑤ 定义 onClick()方法，用户单击 Next 按钮后触发 onNext()方法。在该方法中读取邮件的 URL 地址，URL 地址包含有用户名和密码等信息。程序再跳转到 actionCheckSettings()方法中对用户的设置进行检查。

```
      public void onClick(View v) {
        switch (v.getId()) {
            case R.id.next:
                onNext();
                break;
        }
    }
private void onNext() {
        try {
            URI uri = getUri();
            mAccount.setSenderUri(uri.toString());
        } catch (URISyntaxException use) {
            throw new Error(use);
        }
        //检查用户
        AccountCheckSettings.actionCheckSettings(this, mAccount, false, true);
    }
```



(2) 邮箱发送设置的布局文件是 AccountSetupOutgoing.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
  <ScrollView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout width="fill parent"
   android:layout height="fill parent"
   android:scrollbarStyle="outsideInset">
   <LinearLayout
      android:layout width="fill parent"
      android:layout height="fill parent"
      android:orientation="vertical">
      <TextView
         android:text="@string/account setup outgoing security label"
         android:layout height="wrap content"
         android:layout width="fill parent"
         android:textAppearance="?android:attr/textAppearanceSmall"
         android:textColor="?android:attr/textColorPrimary" />
      <Spinner
         android:id="@+id/account security type"
         android:layout height="wrap content"
         android:layout width="fill parent" />
      <CheckBox
         android:id="@+id/account require login"
         android:layout width="fill parent"
         android:layout height="wrap content"
         android:text="@string/account setup outgoing require login label" />
      <LinearLayout
         android:id="@+id/account require login settings"
         android:layout width="fill parent"
         android:layout_height="fill_parent"
         android:orientation="vertical"
         android:visibility="gone">
      </LinearLayout>
</ScrollView>
```

以上代码中“android:scrollbarStyle="outsideInset"”引用的是系统自带的一个外观，“？”表示系统是否有这种外观，否则使用默认的外观。Android 的系统自带的文字外观设置及实际显示效果图的设置如：textappearancebutton、textappearanceinverse、textappearancelarge、textappearancelargeinverse、textappearancemedium、extappearancesmallinverse、textappearancemediuminverse 和 textappearancesmall。

16.4.6　邮箱用户检查

设置好邮件后，单击 Next 按钮，将弹出邮箱用户检查界面，如图 16-14 所示。



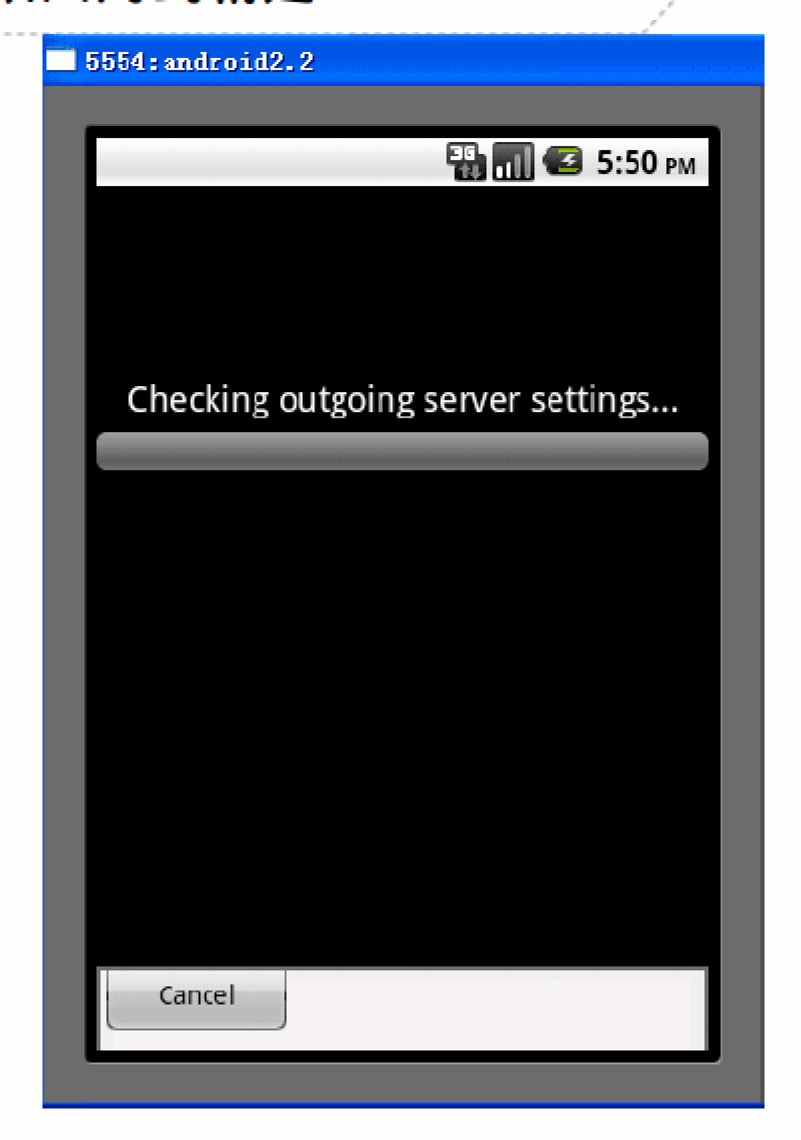


图 16-14　邮箱用户检查界面

(1) 编写文件 AccountCheckSettings.java，在此定义邮箱用户检查界面，主要代码如下。

① 定义 actionCheckSettings()方法做数据初始化操作，startActivityForResult()方法的第一个参数是一个 Intent；第二个参数是返回码。通过方法的不同返回码，可以区分不同的 Activity，当启动了某个 Activity 后，返回码依然关联着当前进程所处理的 Activity。当操作完成后，会有特定的返回值作为响应某些事件。即 Activity 执行 finish()以后执行回调方法 onActivityResult，而使用 startActivity 方法后却不会执行回调。对应代码如下：

```
public class AccountCheckSettings extends Activity implements
OnClickListener {
    private static final String EXTRA ACCOUNT = "account";
    private static final String EXTRA CHECK INCOMING = "checkIncoming";
    private static final String EXTRA CHECK OUTGOING = "checkOutgoing";
    private Handler mHandler = new Handler();
    private ProgressBar mProgressBar;
    private TextView mMessageView;
    private Account mAccount;
    private boolean mCheckIncoming;
    private boolean mCheckOutgoing;
    private boolean mCanceled;
    private boolean mDestroyed;
    public static void actionCheckSettings(Activity context, Account account,
            boolean checkIncoming, boolean checkOutgoing) {
        Intent i = new Intent(context, AccountCheckSettings.class);
       //为 Account 对象的键名赋值
        i.putExtra(EXTRA ACCOUNT, account);
        i.putExtra(EXTRA CHECK INCOMING, checkIncoming);
        i.putExtra(EXTRA CHECK OUTGOING, checkOutgoing);
       //指定将要启动的 Activity 的编号
        context.startActivityForResult(i, 1);
    }
```



② 定义 onCreate 方法初始化窗体，定义了 mProgressBar 控件，该控件主要是作为一个进度条。mProgressBar 对象的 setIndeterminate()方法开启滚动效果，getIntent()方法取得当前 Intent 的信息实例，然后调用各 get 方法获取对应的值。对应代码如下：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity account check settings);
    mMessageView = (TextView)findViewById(R.id.message);
    mProgressBar = (ProgressBar)findViewById(R.id.progress);
    ((Button)findViewById(R.id.cancel)).setOnClickListener(this);
    setMessage(R.string.account setup check settings retr info msg);
    //打开进度条的滚动效果
    mProgressBar.setIndeterminate(true);
    mAccount = (Account)getIntent().getSerializableExtra(EXTRA ACCOUNT);
    mCheckIncoming = (boolean)getIntent().getBooleanExtra
          (EXTRA CHECK INCOMING, false);
    mCheckOutgoing = (boolean)getIntent().getBooleanExtra
          (EXTRA_CHECK_OUTGOING, false);
```

③ 实例化一个线程 Thread，setThreadPriority()方法设置线程在后台执行。程序开始时针对用户的发送邮件进行检查，如果当前 Activity 状态处于 Destroyed 和 Canceled 线程退出。对象 Sender 调用方法 getInstance()读取用户的 URL 信息，然后调用 open()方法打开地址。如果能够顺利地完成一次 close()和 open()方法的操作，并且不报异常，说明用户提供的地址可以正确使用，如图 16-15 所示。对应代码如下：

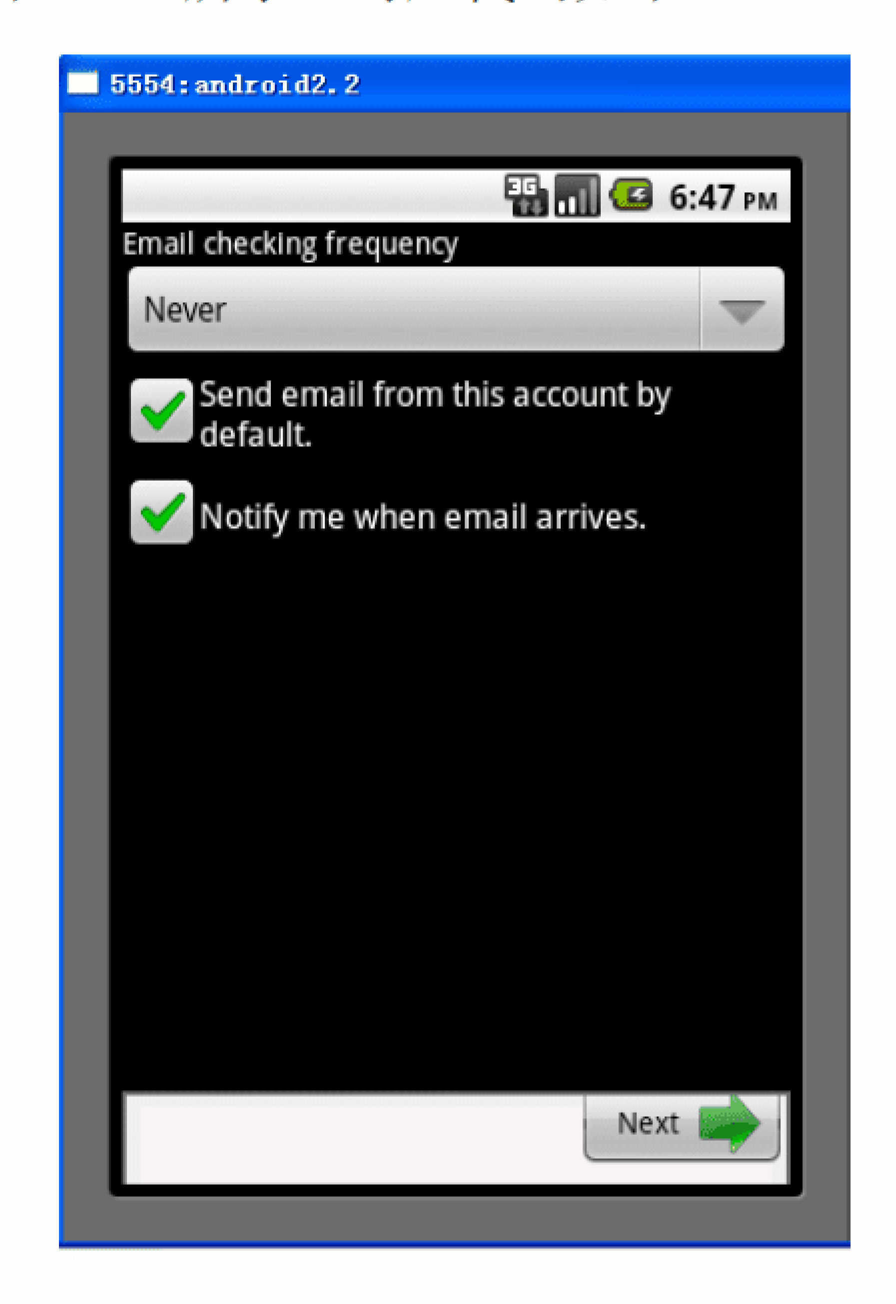


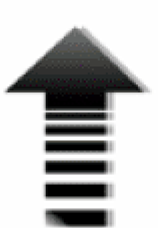
图 16-15　用户检查成功

```
    new Thread() {
        public void run() {
```

```
//设置线程执行级别
        Process.setThreadPriority(Process.THREAD PRIORITY BACKGROUND);
        try {
            if (mDestroyed) {
                return;
            }
            if (mCanceled) {
                finish();
                return;
            }
            if (mCheckIncoming) {
                setMessage(R.string.account setup check settings
                    check incoming msg);
            }
            if (mDestroyed) {
                return;
            }
            if (mCanceled) {
                finish();
                return;
            }
            if (mCheckOutgoing) {
          setMessage(R.string.account setup check settings
                check outgoing msg);
                Sender sender = Sender.getInstance(mAccount.getSenderUri());
                sender.close();
                sender.open();
                sender.close();
            }
            if (mDestroyed) {
                return;
            }
            if (mCanceled) {
                finish();
                return;
            }
            setResult(RESULT OK);
            finish();
```

④ 在此定义捕获 AuthenticationFailedException 类型、CertificateValidationException 类型和 MessagingException 类型的异常。对应代码如下：

```
        } catch (final AuthenticationFailedException afe) {
            String message = afe.getMessage();
            int id = (message == null)
                    ? R.string.account setup failed dlg auth message
                    : R.string.account setup failed dlg auth message fmt;
            //显示错误信息
            showErrorDialog(id, message);
        } catch (final CertificateValidationException cve) {
            String message = cve.getMessage();
```



```
                    int id = (message == null)
                    ? R.string.account setup failed dlg certificate message
                    : R.string.account setup failed dlg certificate message fmt;
                    showErrorDialog(id, message);
                } catch (final MessagingException me) {
                    int id;
                    String message = me.getMessage();
                    switch (me.getExceptionType()) {
                        case MessagingException.IOERROR:
                            id = R.string.account setup failed ioerror;
                            break;
                        case MessagingException.TLS REQUIRED:
                            id = R.string.account setup failed tls required;
                            break;
                        case MessagingException.AUTH REQUIRED:
                            id = R.string.account setup failed auth required;
                            break;
                        case MessagingException.GENERAL SECURITY:
                            id = R.string.account setup failed security;
                            break;
                        default:
                            id = (message == null)
                            ? R.string.account setup failed dlg server message
                            : R.string.account setup failed dlg server message fmt;
                            break;
                    }
                    showErrorDialog(id, message);
                }
            }
        }.start();
    }
```

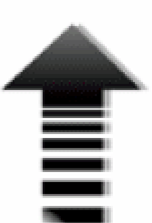
⑤ 定义 showErrorDialog()方法显示错误对话框，具体内容显示在 AlertDialog 实例中指定，同时中止进度条的滚动显示 setIndeterminate(false)，对应代码如下：

```
private void showErrorDialog(final int msgResId, final Object... args) {
        mHandler.post(new Runnable() {
            public void run() {
                if (mDestroyed) {
                    return;
                }
                //关闭进度条的滚动效果
                mProgressBar.setIndeterminate(false);
                //创建一个提示对话框
                new AlertDialog.Builder(AccountCheckSettings.this)
                        .setIcon(android.R.drawable.ic dialog alert)
                        .setTitle(getString(R.string.account setup failed
                            dlg title))
                        .setMessage(getString(msgResId, args))
                        .setCancelable(true)
                        .setPositiveButton( getString(R.string.account_setup_
```

```
                        failed dlg edit details action),
                            new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,
                                    int which) {
                                    finish();
                                }
                            })
                    .show();
            }
        });
    }
  private void onCancel() {
        mCanceled = true;
        setMessage(R.string.account_setup_check_settings_canceling_msg);
    }
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.cancel:
                onCancel();
                break;
        }
    }
```

(2) 设置用户别名的布局文件是 activity_account_setup_incoming.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="fill parent"
    android:layout height="fill parent"
    android:orientation="vertical">
    <View
        android:layout width="fill parent"
        android:layout height="100sp" />
    <TextView
        android:id="@+id/message"
        android:layout height="wrap content"
        android:layout width="fill parent"
        android:gravity="center horizontal"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="?android:attr/textColorPrimary"
        android:paddingBottom="6px" />
    <ProgressBar
        android:id="@+id/progress"
        android:layout height="wrap content"
        android:layout width="fill parent"
        style="?android:attr/progressBarStyleHorizontal" />
    <View
        android:layout width="fill parent"
        android:layout height="0px"
        android:layout weight="1" />
    <RelativeLayout
```



```
        android:layout width="fill parent"
        android:layout height="54dip"
        android:background="@android:drawable/menu full frame">
        <Button
            android:id="@+id/cancel"
            android:text="@string/cancel action"
            android:layout height="wrap content"
            android:layout width="wrap content"
            android:minWidth="100dip"
            android:layout centerVertical="true" />
    </RelativeLayout>
</LinearLayout>
```

以上代码“android:drawableRight="@drawable/button_indicator_next"”中，drawable 在 Android SDK 中的主要作用是：在 XML 中定义各种动画，然后把 XML 当作 drawable 资源来读取，通过 drawable 显示动画。

16.4.7　设置用户别名

在成功通过地址检测后，单击 Next 按钮，将弹出设置用户别名界面，如图 16-16 所示。

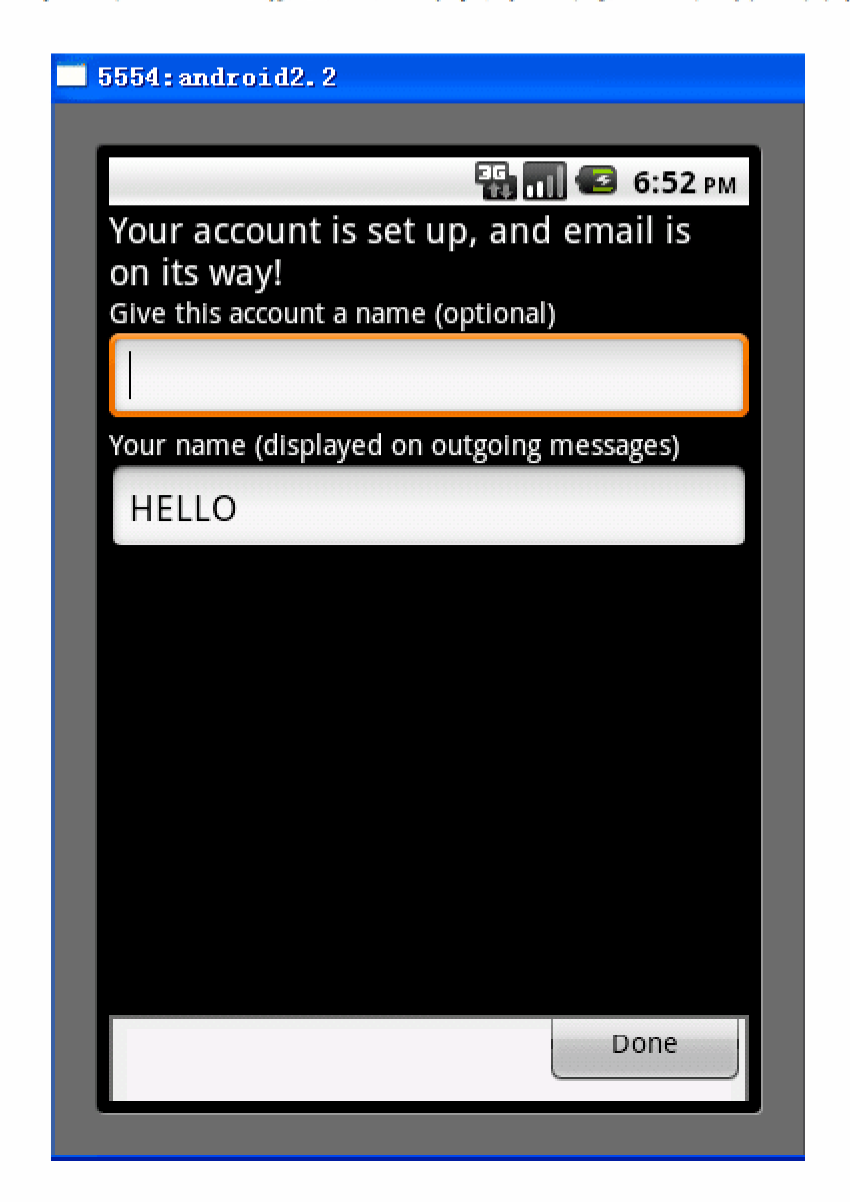


图 16-16　设置用户别名界面

(1) 编写文件 AccountSetupNames.java，在此定义设置用户别名界面，主要代码如下。

① 定义 actionSetNames()方法做数据初始化操作，startActivity()方法启动 Activity；onCreate()方法定义两个文本框组件，还定义 setOnClickListener 监听器；validateFields()方法一旦发现文本框中的内容都发生变化后，执行监听器里面的动作。对应代码如下：

```
public class AccountSetupNames extends Activity implements
OnClickListener {
    private static final String EXTRA ACCOUNT = "account";
    private EditText mDescription;
```

```
    private EditText mName;
    private Account mAccount;
    private Button mDoneButton;
    public static void actionSetNames(Context context, Account account) {
        //为 AccountSetupNames 窗体的 Activity 赋值
        Intent i = new Intent(context, AccountSetupNames.class);
        i.putExtra(EXTRA ACCOUNT, account);
        //启动 Account 窗体
context.startActivity(i);
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity account setup names);
        mDescription = (EditText)findViewById(R.id.account description);
        mName = (EditText)findViewById(R.id.account name);
        mDoneButton = (Button)findViewById(R.id.done);
        //定义监听器
        mDoneButton.setOnClickListener(this);
        TextWatcher validationTextWatcher = new TextWatcher() {
            public void afterTextChanged(Editable s) {
                validateFields();
            }
        };
```

② 调用 mName.setText()方法保存用户名称。onNext()方法将当前设置的用户名保存在 Preferences 对象中，然后 Activity 跳转到邮件编辑界面 EmailCpsActivity。对应代码如下：

```
        mName.addTextChangedListener(validationTextWatcher);
        mName.setKeyListener(TextKeyListener.getInstance(false, Capitalize.WORDS));
        mAccount = (Account)getIntent().getSerializableExtra(EXTRA ACCOUNT);
        // mDescription.setText(mAccount.getDescription());
        if (mAccount.getName() != null) {
            mName.setText(mAccount.getName());
        }
        if (!Utility.requiredFieldValid(mName)) {
            mDoneButton.setEnabled(false);
        }
    }
    private void validateFields() {
        mDoneButton.setEnabled(Utility.requiredFieldValid(mName));
        Utility.setCompoundDrawablesAlpha(mDoneButton,
           mDoneButton.isEnabled() ? 255 : 128);
    }
    private void onNext() {
        if (Utility.requiredFieldValid(mDescription)) {
            mAccount.setDescription(mDescription.getText().toString());
        }
        mAccount.setName(mName.getText().toString());
        mAccount.save(Preferences.getPreferences(this));
        //定义一个 Activity 名为 EmailCpsActivity
```

```
        Intent intent = new Intent(this, EmailCpsActivity.class);
        intent.putExtra(EXTRA ACCOUNT , mAccount);
        //启动 Activity
        startActivity(intent);
        //执行
        finish();
    }
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.done:
                onNext();
                break;
        }
    }
}
```

(2) 设置用户别名的布局文件是 activity_account_setup_names.xml，主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="fill parent"
    android:layout height="fill parent"
    android:orientation="vertical">
    <TextView
        android:text="@string/account setup names instructions"
        android:layout height="wrap content"
        android:layout width="fill parent"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="?android:attr/textColorPrimary" />
    <TextView
        android:text="@string/account setup names account name label"
        android:layout_height="wrap_content"
        android:layout width="fill parent"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="?android:attr/textColorPrimary" />
    <EditText
        android:id="@+id/account description"
        android:inputType="textCapWords"
        android:imeOptions="actionDone"
        android:layout height="wrap content"
        android:layout width="fill parent" />
    <TextView
        android:text="@string/account setup names user name label"
        android:layout height="wrap content"
        android:layout width="fill parent"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="?android:attr/textColorPrimary" />
    <EditText
        android:id="@+id/account name"
        android:inputType="textPersonName"
        android:imeOptions="actionDone"
```

```
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />
    <View
        android:layout_height="0px"
        android:layout_width="fill_parent"
        android:layout_weight="1" />
    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="54dip"
        android:background="@android:drawable/menu_full_frame">
        <Button
            android:id="@+id/done"
            android:text="@string/done_action"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:minWidth="100dip"
            android:layout_alignParentRight="true"
            android:layout_centerVertical="true" />
    </RelativeLayout>
</LinearLayout>
```

在以上代码“android:inputType="textCapWords"”中，inputType 设置键盘类型。textcapcharacters 代表字母大小，numbersigned 代表有符号数字格式，textcapwords 代表单词首字母大小，textcapsentences 代表仅第一个字母大小，textautocomplete 代表自动完成，textmultiline 代表多行输入，textimemultiline 代表输入法多行，textnosuggestions 代表不提示，textemailaddress 代表电子邮件地址，textemailsubject 代表邮件主题，textshortmessage 代表短信息，textpersonname 代表人名，textpostaladdress 代表地址，textpassword 代表密码，textvisiblepassword 代表可见密码，textwebedittext 代表作为网页表单的文本，textfilte 代表文本筛选过滤，textphonetic 代表拼音输入。

16.4.8 用户邮件编辑

在设置完别名后，单击 Next 按钮，将弹出用户邮件编辑界面，如图 16-17 所示。

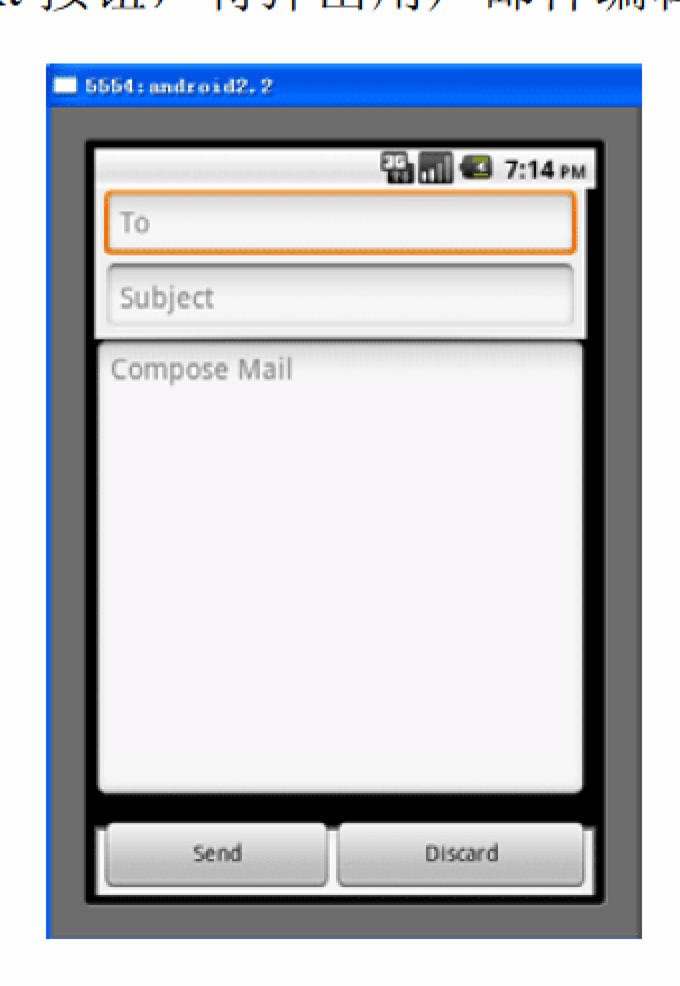
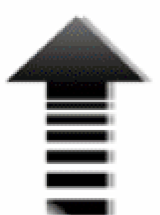


图 16-17 用户邮件编辑界面



(1) 编写文件 EmailCpsActivity.java，在此定义用户邮件编辑界面，主要代码如下。

① 定义 Handler 一个实例化对象，每一个 Handler 类都和一个唯一的线程(以及这个线程的 MessageQueue)关联，向它所关联的 MessageQueue 递送 Messages/Runnables。它的主要用途：按计划发送消息或执行某个 Runnable(使用 Post 方法)；从其他线程中发送来的消息放入消息队列中，避免线程冲突(常见于更新 UI 线程)。对应代码如下：

```
public class EmailCpsActivity extends Activity implements
OnClickListener, OnFocusChangeListener {
    private static final String EXTRA_ACCOUNT = "account";
    private static final int MSG_PROGRESS_ON = 1;
    private static final int MSG_PROGRESS_OFF = 2;
    private static final int MSG_UPDATE_TITLE = 3;
    private static final int MSG_SKIPPED_ATTACHMENTS = 4;
    private static final int MSG_SAVED_DRAFT = 5;
    private static final int MSG_DISCARDED_DRAFT = 6;
    private Account mAccount;
    private MultiAutoCompleteTextView mToView;
    private MultiAutoCompleteTextView mCcView;
    private MultiAutoCompleteTextView mBccView;
    private EditText mSubjectView;
    private EditText mMessageContentView;
    private Button mSendButton;
    private Button mDiscardButton;
    private ProgressDialog progress;
    private Handler mHandler = new Handler() {
        @Override
        public void handleMessage(android.os.Message msg) {
            switch (msg.what) {
                case MSG_PROGRESS_ON:
               //开启滚动条的滚动效果
                    setProgressBarIndeterminateVisibility(true);
                    break;
                case MSG_PROGRESS_OFF:
                //关闭滚动条的滚动效果
                    setProgressBarIndeterminateVisibility(false);
                    break;
                case MSG_UPDATE_TITLE:
                    updateTitle();
                    break;
                case MSG_SKIPPED_ATTACHMENTS:
                    Toast.makeText(
                            EmailCpsActivity.this, getString(R.string.message
                                _compose_attachments_skipped_toast),
                            Toast.LENGTH_LONG).show();
                    break;
                case MSG_SAVED_DRAFT:
                    Toast.makeText(
                            EmailCpsActivity.this,
                            getString(R.string.message_saved_toast),
                            Toast.LENGTH_LONG).show();
```

```
                break;
            case MSG_DISCARDED_DRAFT:
                Toast.makeText(
                        EmailCpsActivity.this,
                        getString(R.string.message_discarded_toast),
                        Toast.LENGTH_LONG).show();
                break;
            default:
                super.handleMessage(msg);
                break;
        }
    }
};
private Validator mAddressValidator;
@Override
```

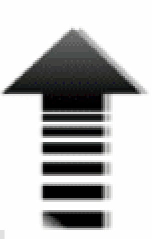
② 定义onCreate()方法做数据初始化操作，定义一个MultiAutoCompleteTextView对象，该对象继承自AutoCompleteTextView的可编辑文本视图，能够对用户输入的文本进行有效地扩充提示，而不需要用户输入整个内容。

③ requestWindowFeature()方法的功能是启用窗体的扩展特性。参数是 Window 类中定义的常量。DEFAULT_FEATURES 为系统默认状态，一般不需要指定；FEATURE_CONTEXT_MENU 为启用 ContextMenu，默认该项已启用，一般无须指定；FEATURE_CUSTOM_TITLE 为自定义标题。当需要自定义标题时必须指定。例如：标题是一个按钮时；FEATURE_INDETERMINATE_PROGRESS 为不确定的进度；FEATURE_LEFT_ICON 为标题栏左侧的图标为 FEATURE_NO_TITLE 为无标题；FEATURE_OPTIONS_PANEL 为启用"选项面板"功能，默认已启用；FEATURE_PROGRESS 为进度指示器功能；FEATURE_RIGHT_ICON 为标题栏右侧的图标。对应代码如下：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
    //加载 activity_compose.xml 布局界面文件
    setContentView(R.layout.activity_compose);
    mAddressValidator = new EmailAddressValidator();
    mToView = (MultiAutoCompleteTextView)findViewById(R.id.to);
    mCcView = (MultiAutoCompleteTextView)findViewById(R.id.cc);
    mBccView = (MultiAutoCompleteTextView)findViewById(R.id.bcc);
    mSubjectView = (EditText)findViewById(R.id.subject);
    mMessageContentView = (EditText)findViewById(R.id.message_content);
    mSendButton = (Button)findViewById(R.id.send);
    mDiscardButton = (Button)findViewById(R.id.discard);
```

④ 定义InputFilter对象的实例recipientFilter 使用输入过滤器InputFilter约束用户输入。定义一个返回类型为CharSequence的方法filter检查用户的所有输入是否合法。对应代码如下：

```
    InputFilter recipientFilter = new InputFilter() {
    //定义字符过滤方法
```



```
public CharSequence filter(CharSequence source, int start, int end,
    Spanned dest,
                int dstart, int dend) {
            if (end-start != 1 || source.charAt(start) != ' ') {
                return null;
            }
            int scanBack = dstart;
            boolean dotFound = false;
            while (scanBack > 0) {
                char c = dest.charAt(--scanBack);
                switch (c) {
                    case '.':
                        dotFound = true;    // one or more dots are req'd
                        break;
                    case ',':
                        return null;
                    case '@':
                        if (!dotFound) {
                            return null;
                        }
                        if (source instanceof Spanned) {
                            SpannableStringBuilder sb = new
                                SpannableStringBuilder(",");
                            sb.append(source);
                            return sb;
                        } else {
                            return ", ";
                        }
                    default:
                        // just keep going
                }
            }
            // no termination cases were found, so don't edit the input
            return null;
        }
    };
    InputFilter[] recipientFilters = new InputFilter[] { recipientFilter };
    // NOTE: assumes no other filters are set
```

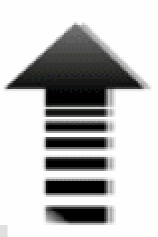
⑤ 将输入过滤器作用于 mToView、mCcView、mBccView、mToView、mToView 和 mCcView 组件之上。setOnFocusChangeListener 方法将焦点定位于该组件上，为“发送”和“取消”按钮定义 setOnClickListener 监听器。对应代码如下：

```
   //为组件指定过滤规则
    mToView.setFilters(recipientFilters);
    mCcView.setFilters(recipientFilters);
    mBccView.setFilters(recipientFilters);
    mToView.setTokenizer(new Rfc822Tokenizer());
    mToView.setValidator(mAddressValidator);
    mCcView.setTokenizer(new Rfc822Tokenizer());
    mCcView.setValidator(mAddressValidator);
```

```
        mBccView.setTokenizer(new Rfc822Tokenizer());
        mBccView.setValidator(mAddressValidator);
        mSendButton.setOnClickListener(this);
        mDiscardButton.setOnClickListener(this);
        mSubjectView.setOnFocusChangeListener(this);
        Intent intent = getIntent();
        mAccount = (Account) intent.getSerializableExtra(EXTRA_ACCOUNT);
        updateTitle();
    }
    @Override
    //暂停后恢复调用
    public void onResume() {
        super.onResume();
    }
    @Override
    //暂停
    public void onPause() {
        super.onPause();
    }
    @Override
    //退出
    public void onDestroy() {
        super.onDestroy();
    }
    private void updateTitle() {
        if (mSubjectView.getText().length() == 0) {
            setTitle(R.string.compose_title);
        } else {
            setTitle(mSubjectView.getText().toString());
        }
    }
    //焦点发生变化
    public void onFocusChange(View view, boolean focused) {
        if (!focused) {
            updateTitle();
        }
    }
    private Address[] getAddresses(MultiAutoCompleteTextView view) {
        Address[] addresses = Address.parse(view.getText().toString().trim());
        return addresses;
    }
```

⑥ 定义一个 MimeMessage 对象 message 实例，类 MimeMessage 继承自定义类 Message，封装邮件发送时的信息。调用 setFrom()方法、setRecipients()方法和 setSubject()方法进行邮件信息头封装操作。定义一个 TextBody 对象 body 实例，调用 setBody()方法将文本内容写入其中。对应代码如下：

```
    private MimeMessage createMessage() throws MessagingException {
        MimeMessage message = new MimeMessage();
        message.setSentDate(new Date());
        Address from = new Address(mAccount.getEmail(), mAccount.getName());
```

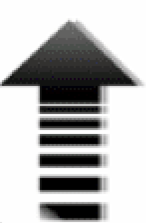


```
        message.setFrom(from);
        message.setRecipients(RecipientType.TO, getAddresses(mToView));
        message.setRecipients(RecipientType.CC, getAddresses(mCcView));
        message.setRecipients(RecipientType.BCC, getAddresses(mBccView));
        message.setSubject(mSubjectView.getText().toString());
        String text = mMessageContentView.getText().toString();
      //输出日志信息
        Log.d(Email.LOG TAG, text);
        TextBody body = new TextBody(text);
        message.setBody(body);
        return message;
    }
    private void sendMessage() {
    //定义一个进度条
     progress = ProgressDialog.show(this, "", "sending...");
        final MimeMessage message;
        try {
            message = createMessage();
        }
        catch (MessagingException me) {
  //打出日志
            Log.e(Email.LOG TAG, "Failed to create new message for send or
                save.", me);
            throw new RuntimeException("Failed to create a new message for
                send or save.", me);
        }
        Thread thread = new Thread(new Runnable(){
              @Override
              public void run() {
                  try {
                  Sender sender = Sender.getInstance
                      (mAccount.getSenderUri());
                  ArrayList<Part> viewables = new ArrayList<Part>();
                    ArrayList<Part> attachments = new ArrayList<Part>();
                    MimeUtility.collectParts(message, viewables, attachments);
                    StringBuffer sbHtml = new StringBuffer();
                    StringBuffer sbText = new StringBuffer();
                    for (Part viewable : viewables) {
                        try {
                            String text = MimeUtility.getTextFromPart(viewable);
                            /*
                        * Anything with MIME type text/html will be stored
                            as such. Anything
                             * else will be stored as text/plain.
                             */
                            if (viewable.getMimeType().equalsIgnoreCase
                                ("text/html")) {
                                sbHtml.append(text);
                            }
                            else {
                                sbText.append(text);
```

```
                    }
                } catch (Exception e) {
            throw new MessagingException("Unable to get text for
                message part", e);
                }
            }
            message.setUid("email" + UUID.randomUUID().toString());
        message.setHeader(MimeHeader.HEADER CONTENT TYPE, "multipart/mixed");
            MimeMultipart mp = new MimeMultipart();
            mp.setSubType("mixed");
            message.setBody(mp);
              String htmlContent = sbHtml.toString();
              String textContent = sbText.toString();
              if (htmlContent != null) {
                  TextBody body = new TextBody(htmlContent);
                  MimeBodyPart bp = new MimeBodyPart(body, "text/html");
                  mp.addBodyPart(bp);
                  Log.v(Email.LOG TAG, htmlContent);
              }
              if (textContent != null) {
                  TextBody body = new TextBody(textContent);
                  MimeBodyPart bp = new MimeBodyPart(body, "text/plain");
                  mp.addBodyPart(bp);
                  Log.v(Email.LOG TAG, textContent);
              }
        //发送信息
            sender.sendMessage(message);
            } catch (MessagingException e) {
               e.printStackTrace();
            }
            progress.dismiss();
            finish();
        }
    });
    thread.start();
}
```

⑦ 对象 Toast 是 Android 中用来显示信息的一种机制，它和 Dialog 有所不同，Toast 没有焦点，而且显示的时间很短，在一定的时间后就会自动消失。sendMessage()正式进行邮件发送操作。对应代码如下：

```
private void onSend() {
    if (getAddresses(mToView).length == 0 &&
            getAddresses(mCcView).length == 0 &&
            getAddresses(mBccView).length == 0) {
        mToView.setError(getString(R.string.message compose error
            no recipients));
        Toast.makeText(this, getString(R.string.message compose error
            no recipients),
                Toast.LENGTH LONG).show();
        return;
```



```
        }
        sendMessage();
    }
    //发送信息主方法
    private void onDiscard() {
        mHandler.sendEmptyMessage(MSG DISCARDED DRAFT);
        finish();
    }
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.send:
                onSend();
                break;
            case R.id.discard:
                onDiscard();
                break;
        }
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.send:
                onSend();
                break;
            case R.id.discard:
                onDiscard();
                break;
            default:
                return super.onOptionsItemSelected(item);
        }
        return true;
    }
}
```

(2) 用户邮件编辑的布局文件是 activity_compose.xml，主要代码如下。

- 以下代码中 android:scrollbarStyle="outsideInset"设置滚动条的风格，
- android:fillViewport="true" ，当定义 scrollview 的子控件不足 scrollbarStyle 大小时，对其设定 fill_parent 属性时不起作用，此时必须加 fillviewport 属性。对应代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout height="fill parent" android:layout width="fill parent"
    android:orientation="vertical">
    <ScrollView android:layout width="fill parent"
        android:layout height="wrap content" android:layout weight="1"
        android:scrollbarStyle="outsideInset"
        android:fillViewport="true">
        <LinearLayout android:orientation="vertical"
            android:layout_width="fill_parent"
```

```
android:layout height="wrap content">
          <LinearLayout android:orientation="vertical"
             android:layout width="fill parent"
             android:layout height="wrap content" android:background="#ededed">
             <MultiAutoCompleteTextView
                android:id="@+id/to" android:layout width="fill parent"
                android:layout height="wrap content"
                android:textAppearance="?android:attr/textAppearanceMedium"
                android:textColor="?android:attr/textColorSecondaryInverse"
                android:layout marginLeft="6px"
                android:layout marginRight="6px"
                android:inputType="textEmailAddress|textMultiLine"
                android:imeOptions="actionNext"
                android:hint="@string/message_compose_to_hint" />
```

- android:hint="@string/message_compose_bcc_hint"，设置 EditText 为空时提示信息。
- android:visibility="gone"，表示此视图是否显示。三个属性分别是：visible 显示；invisible 显示黑背景条；gone 不显示。对应代码如下：

```
<MultiAutoCompleteTextView
   android:id="@+id/cc" android:layout width="fill parent"
   android:layout height="wrap content"
   android:textAppearance="?android:attr/textAppearanceMedium"
   android:textColor="?android:attr/textColorSecondaryInverse"
   android:layout marginLeft="6px"
   android:layout marginRight="6px"
   android:inputType="textEmailAddress|textMultiLine"
   android:imeOptions="actionNext"
   android:hint="@string/message compose cc hint"
   android:visibility="gone" />
<MultiAutoCompleteTextView
   android:id="@+id/bcc" android:layout width="fill parent"
   android:layout height="wrap content"
   android:textAppearance="?android:attr/textAppearanceMedium"
   android:textColor="?android:attr/textColorSecondaryInverse"
   android:layout marginLeft="6px"
   android:layout marginRight="6px"
   android:inputType="textEmailAddress|textMultiLine"
   android:imeOptions="actionNext"
   android:hint="@string/message compose bcc hint"
   android:visibility="gone" />
```

- android:inputType="textEmailSubject|textAutoCorrect|textCapSentences|textImeMultiLine"，设置键盘输入类型。多种类型同时定义时用“｜”分隔符。
- android:gravity="top"，设置控件上信息的位置。参数有 center(居中)、bottom(下)、top(上)、right(右)和 left(左)，如定义左下的效果 android:gravity=" left| bottom "。对应代码如下：

```
             <EditText android:id="@+id/subject"
                android:layout_width="fill_parent"
```



```
                android:textAppearance="?android:attr/textAppearanceMedium"
                android:layout_height="wrap_content"
                android:textColor="?android:attr/textColorSecondaryInverse"
                android:layout_marginLeft="6px"
                android:layout_marginRight="6px"
                android:hint="@string/message_compose_subject_hint"
                android:inputType="textEmailSubject|textAutoCorrect|
                   textCapSentences|textImeMultiLine"
                android:imeOptions="actionNext"
                />
            <LinearLayout android:id="@+id/attachments"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical" />
            <View android:layout_width="fill_parent"
                android:layout_height="1px"
                android:background="@drawable/divider_horizontal_email" />
        </LinearLayout>
        <EditText android:id="@+id/message_content"
            android:textColor="?android:attr/textColorSecondaryInverse"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1.0"
            android:gravity="top"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:hint="@string/message_compose_body_hint"
            android:inputType="textMultiLine|textAutoCorrect|textCapSentences"
            android:imeOptions="actionDone|flagNoEnterAction"
            />
    </LinearLayout>
</ScrollView>
```

- android:paddingTop="5dip"，padding 是站在父组件 view 的角度指定空间位置，规定里面的内容必须与这个父 view 边界的距离。margin 则是根据控件自身指定空间位置，设定其他(上、下、左、右)的 view 之间的距离。对应代码如下：

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:paddingTop="5dip"
    android:paddingLeft="4dip"
    android:paddingRight="4dip"
    android:paddingBottom="1dip"
    android:background="@android:drawable/menu_full_frame" >
    <Button
        android:id="@+id/send"
        android:text="@string/send_action"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:layout_weight="1" />
```

```
        <Button
            android:id="@+id/discard"
            android:text="@string/discard action"
            android:layout height="fill parent"
            android:layout_width="wrap_content"
            android:layout weight="1" />
    </LinearLayout>
</LinearLayout>
```

16.5　打包、签名和发布

当一个 Android 项目开发完毕后，需要打包和签名处理，这样才能放到手机中使用，当然也可以发布到 Market 上去赚钱。下面开始讲解打包、签名、发布 Android 程序的具体过程。

16.5.1　申请会员

申请会员即去 Market 申请成为会员，具体流程如下。

(1) 登录 http://market.android/publish/signup，如图 16-18 所示。

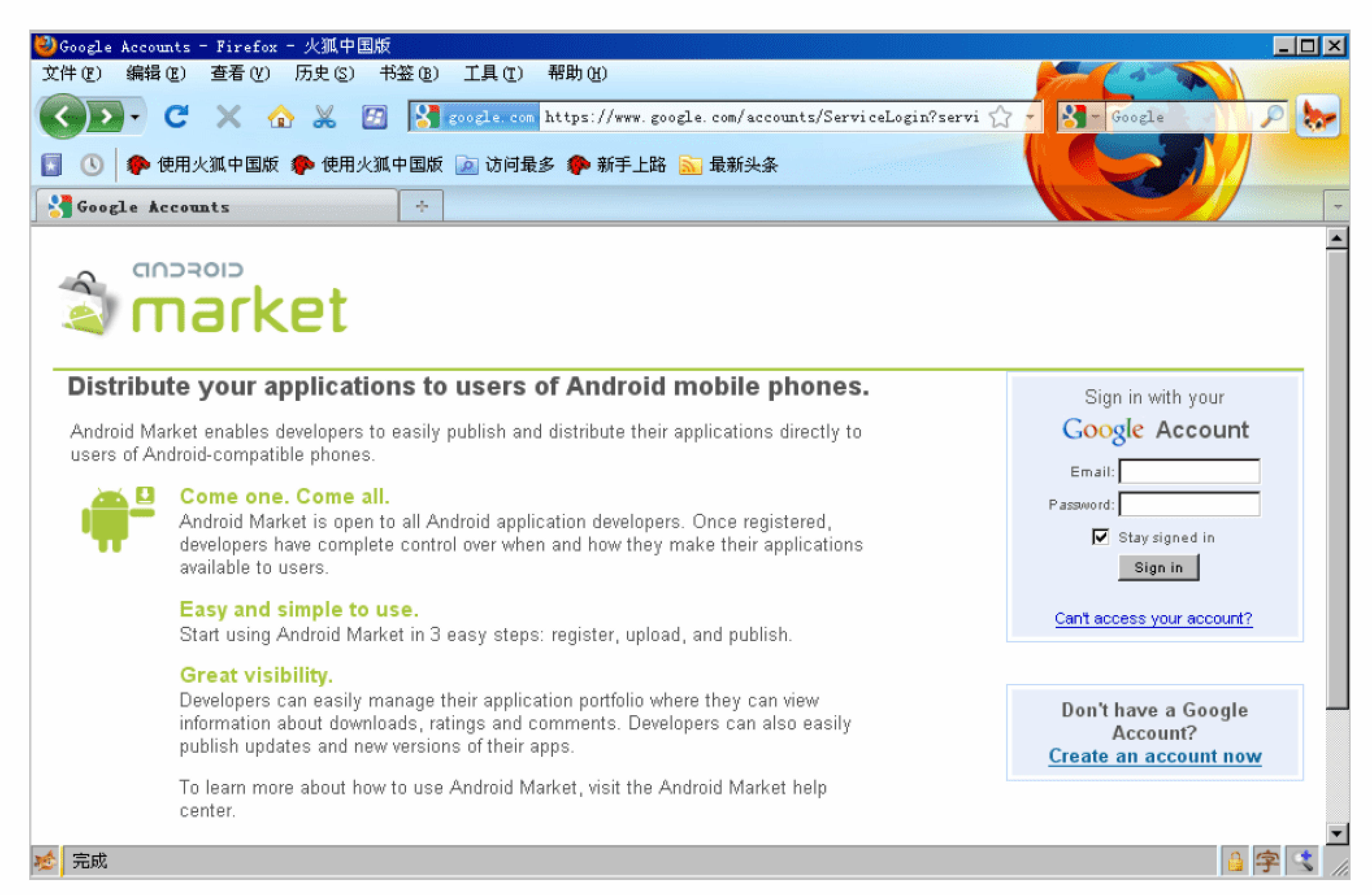
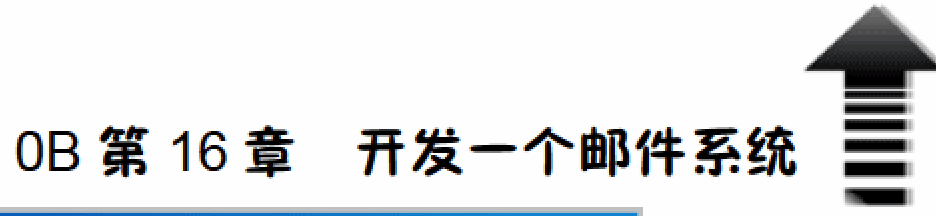


图 16-18　登录Market

(2) 单击 Create an account now 链接，来到注册页面，如图 16-19 所示。



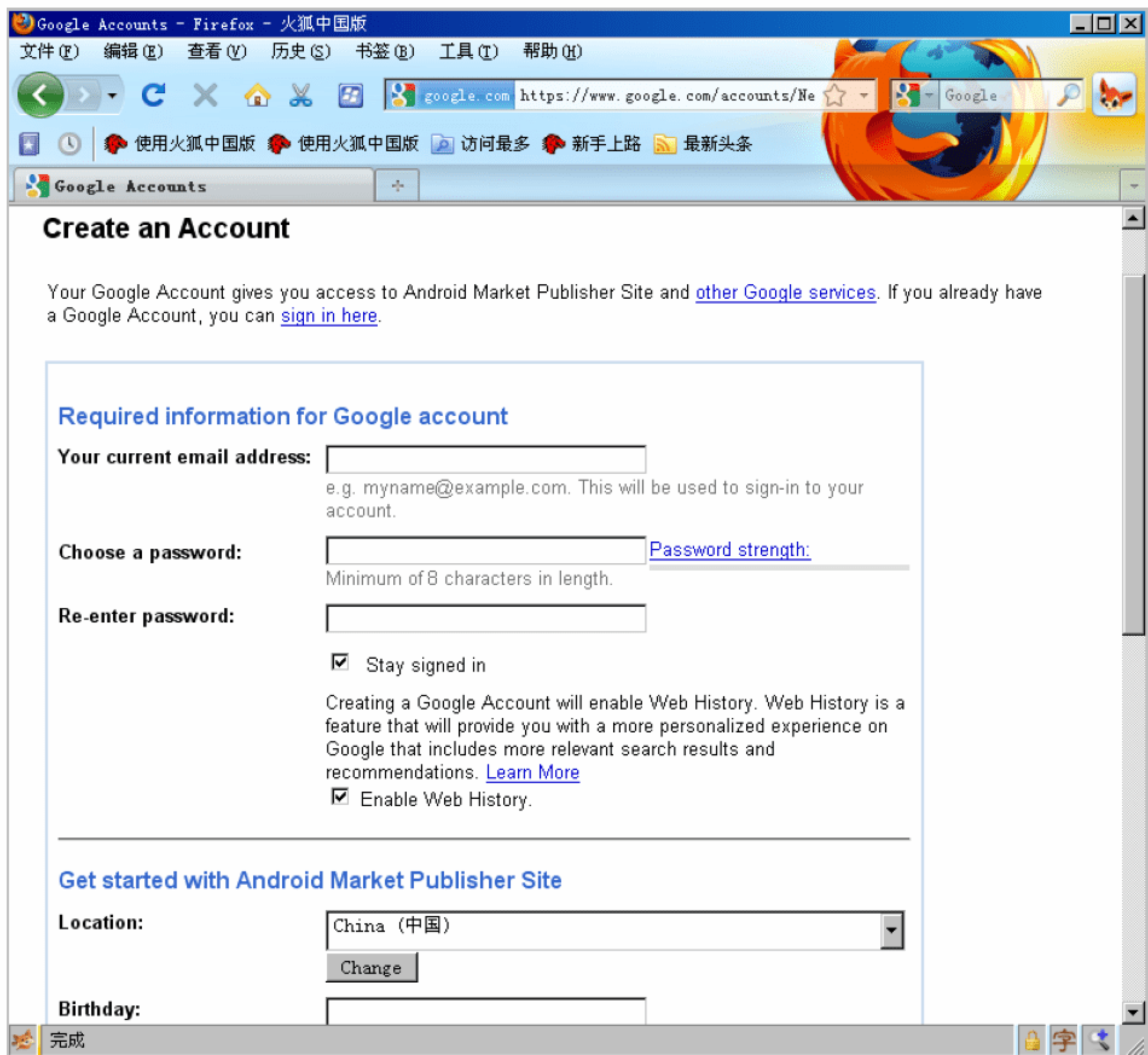


图 16-19　注册页面

(3) 单击同意协议按钮后进入下一步的页面，在此输入手机号码，如图 16-20 所示。

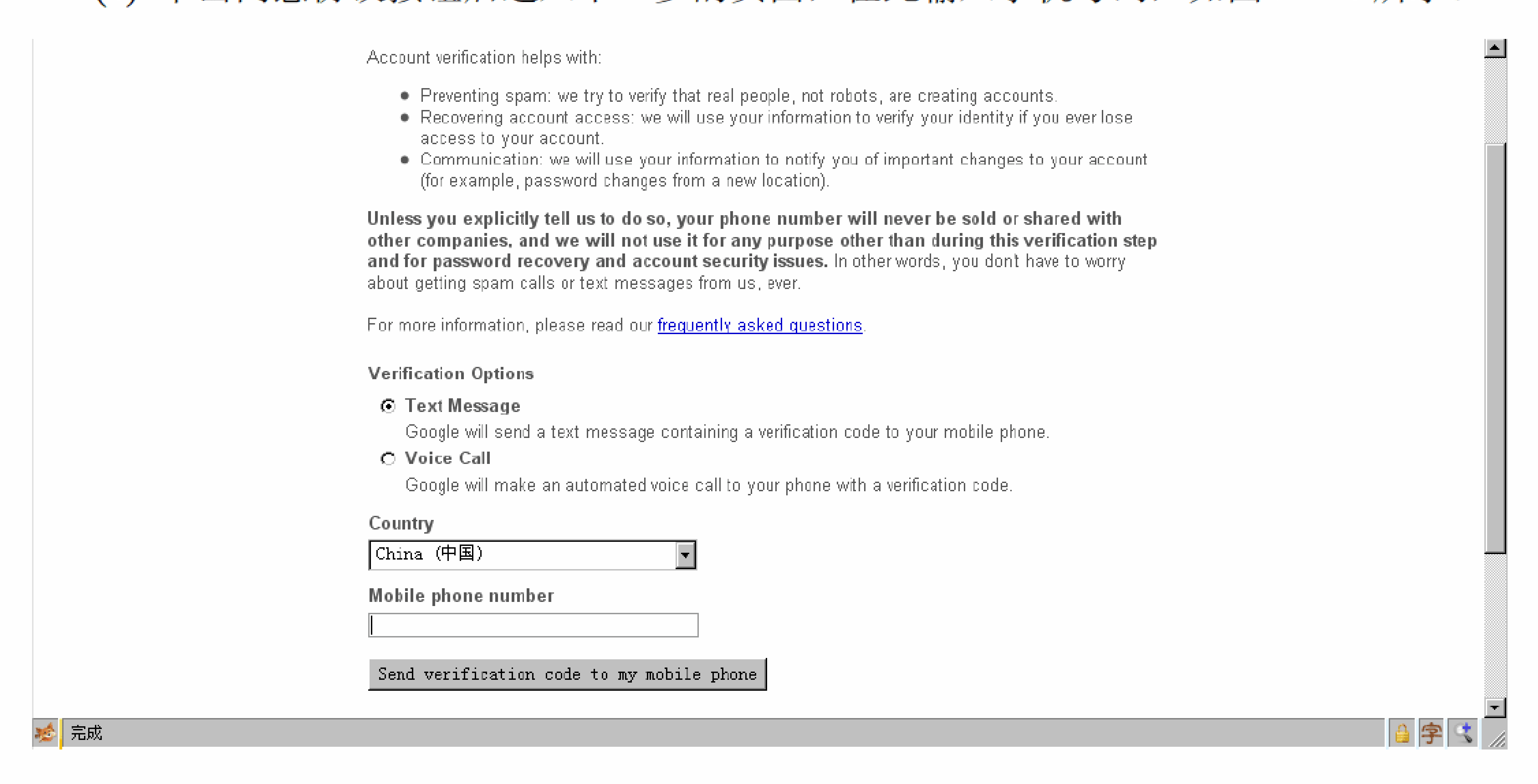


图 16-20　输入手机号码

(4) 在新打开的页面中输入手机获取的验证码，如图 16-21 所示。



图 16-21　输入验证码

(5) 验证通过后，在新打开的页面中继续输入信息，如图 16-22 所示。

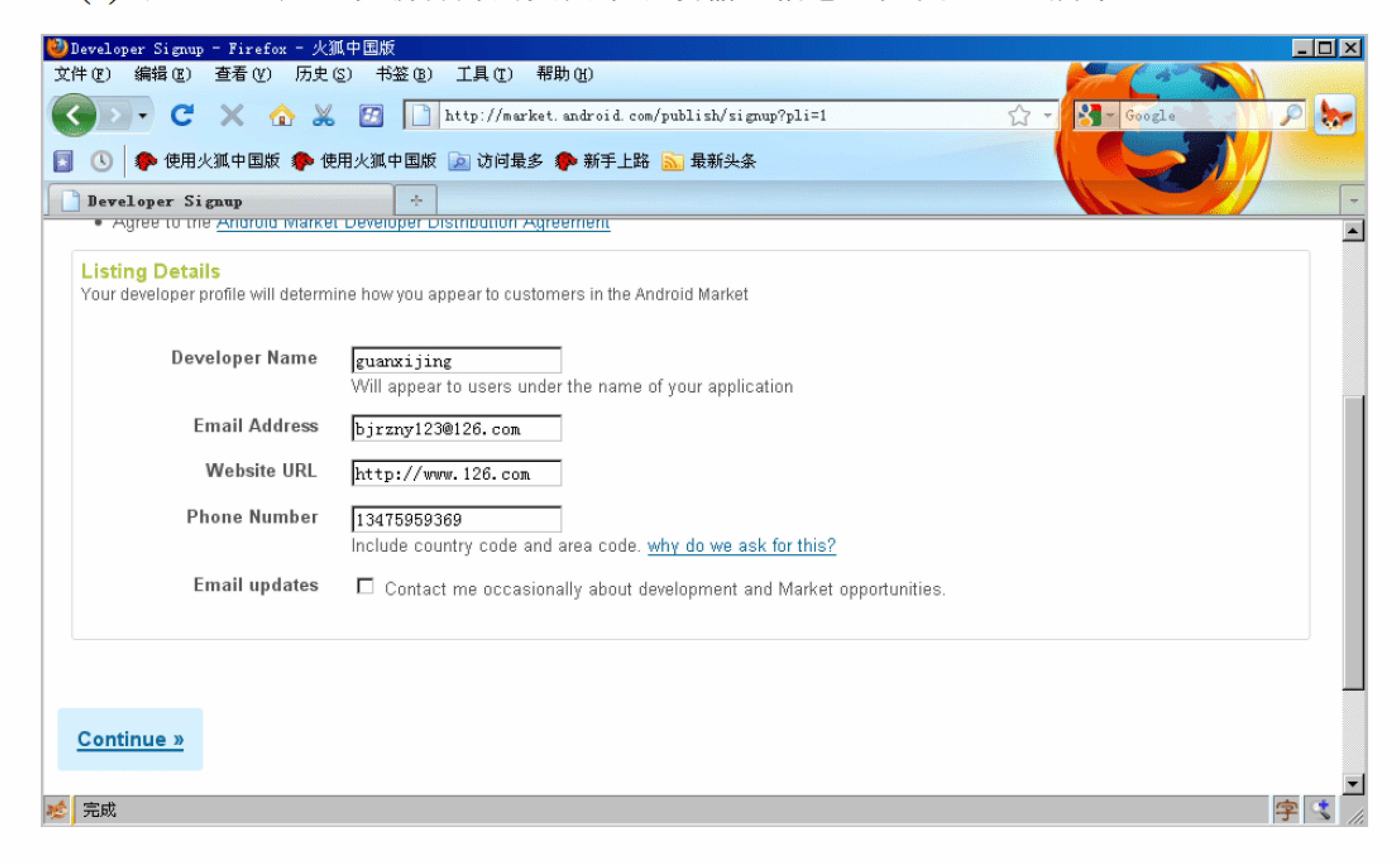


图 16-22　输入信息

(6) 单击 Continue 按钮后，提示需要花费 25 美元，支付后才能成为正式会员，如图 16-23 所示。

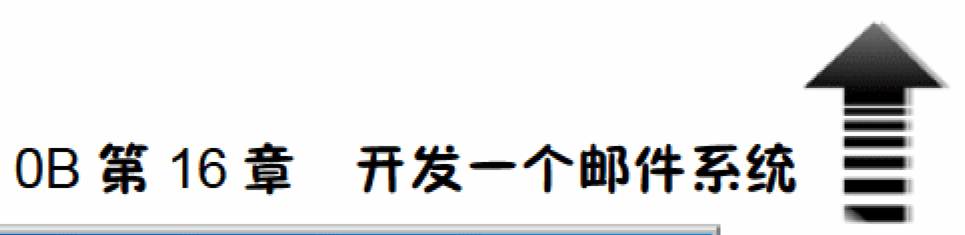




图 16-23 提示支付页面

(7) 单击 Google Checkout Fast checkout through Google 按钮来到支付界面，如图 16-24 所示。

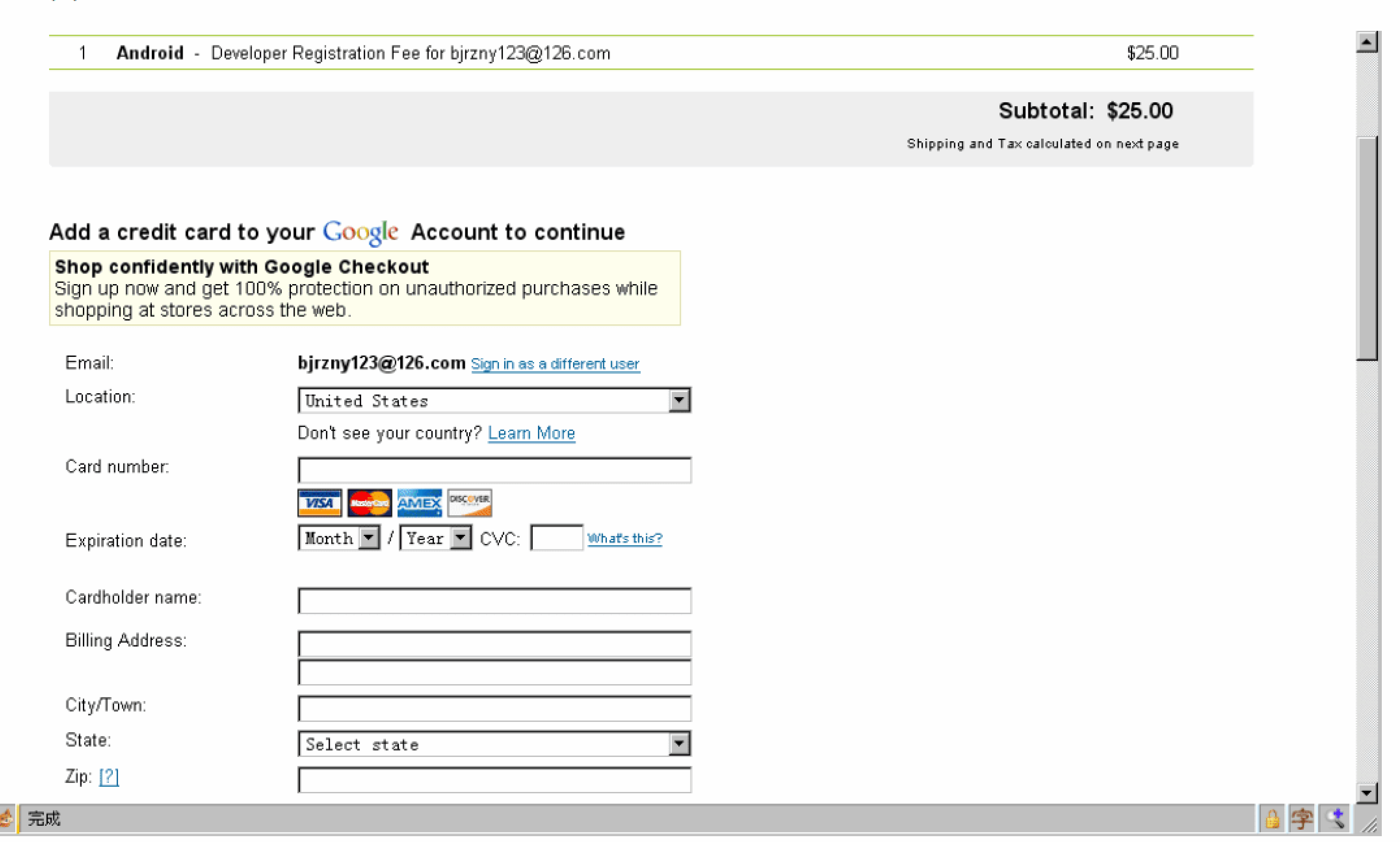


图 16-24 支付界面

在此输入你的信用卡信息，完成支付后即可成为正式会员。

16.5.2 生成签名文件

Android 程序的签名和 Symbian 类似，都可以自签名(Self-signed)，但是在 Android 平

台中，证书初期还显得形同虚设，平时开发时通过 ADB 接口上传的程序会自动被签有 Debug 权限的程序。在需要签名验证上传程序到 Android Market 上时大家都已经发现这个问题了。

Android 签名文件的制作方法有以下两种。

1. 命令行生成

具体流程如下。

(1) 执行如下 cmd 命令：

```
keytool -genkey -alias android123.keystore -keyalg RSA -validity 20000 -keystore android123.keystore
```

然后依次提示用户输入以下信息：

输入 keystore 密码：[密码不回显]

再次输入新密码：[密码不回显]

您的名字与姓氏是什么？

[Unknown]：android123

您的组织单位名称是什么？

[Unknown]：www.android123.com.cn

您的组织名称是什么？

[Unknown]：www.android123.com.cn

您的组织名称是什么？

[Unknown]：www.android123.com.cn

您所在的城市或区域名称是什么？

[Unknown]：New York

您所在的州或省份名称是什么？

[Unknown]：New York

该单位的两字母国家代码是什么

[Unknown]：CN

CN=android123, OU=www.android123.com.cn, O=www.android123.com.cn, L=New York, ST=New York, C=CN 正确吗？

[否]：Y

输入<android123.keystore>的主密码(如果和 keystore 密码相同，按回车)：

其中参数-validity 为证书有效天数，这里我们写 200 天。还有在输入密码时没有回显，只管输入就可以了，一般位数建议使用 20 位，最后需要记下来后面还要用。接下来就可以为 apk 文件签名了。

(2) 执行：

```
jarsigner -verbose -keystore android123.keystore -signedjar android123_signed.apk android123.apk android123.keystore
```

这样就可以生成签名的 apk 文件，假设输入文件 android123.apk，则最终生成 android123_signed.apk 为 Android 签名后的 APK 执行文件。



注意：keytool 用法和 jarsigner 的用法总结。

(1) keytool 用法：

```
-certreq       [-v] [-protected]
                 [-alias <别名>] [-sigalg <sigalg>]
                 [-file <csr_file>] [-keypass <密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-changealias [-v] [-protected] -alias <别名> -destalias <目标别名>
                 [-keypass <密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-delete        [-v] [-protected] -alias <别名>
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-exportcert [-v] [-rfc] [-protected]
                 [-alias <别名>] [-file <认证文件>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-genkeypair [-v] [-protected]
                 [-alias <别名>]
                 [-keyalg <keyalg>] [-keysize <密钥大小>]
                 [-sigalg <sigalg>] [-dname <dname>]
                 [-validity <valDays>] [-keypass <密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]
```

```
-genseckey     [-v] [-protected]
               [-alias <别名>] [-keypass <密钥库口令>]
               [-keyalg <keyalg>] [-keysize <密钥大小>]
               [-keystore <密钥库>] [-storepass <存储库口令>]
               [-storetype <存储类型>] [-providername <名称>]
               [-providerclass <提供方类名称> [-providerarg <参数>]] ...
               [-providerpath <路径列表>]

-help
-importcert [-v] [-noprompt] [-trustcacerts] [-protected]
               [-alias <别名>]
               [-file <认证文件>] [-keypass <密钥库口令>]
               [-keystore <密钥库>] [-storepass <存储库口令>]
               [-storetype <存储类型>] [-providername <名称>]
               [-providerclass <提供方类名称> [-providerarg <参数>]] ...
               [-providerpath <路径列表>]

-importkeystore [-v]
               [-srckeystore <源密钥库>] [-destkeystore <目标密钥库>]
               [-srcstoretype <源存储类型>] [-deststoretype <目标存储类型>]
               [-srcstorepass <源存储库口令>] [-deststorepass <目标存储库口令>]
               [-srcprotected] [-destprotected]
               [-srcprovidername <源提供方名称>]
               [-destprovidername <目标提供方名称>]
               [-srcalias <源别名> [-destalias <目标别名>]
                 [-srckeypass <源密钥库口令>] [-destkeypass <目标密钥库口令>]]
               [-noprompt]
               [-providerclass <提供方类名称> [-providerarg <参数>]] ...
               [-providerpath <路径列表>]

-keypasswd     [-v] [-alias <别名>]
               [-keypass <旧密钥库口令>] [-new <新密钥库口令>]
               [-keystore <密钥库>] [-storepass <存储库口令>]
               [-storetype <存储类型>] [-providername <名称>]
               [-providerclass <提供方类名称> [-providerarg <参数>]] ...
               [-providerpath <路径列表>]

-list          [-v | -rfc] [-protected]
```

[-alias <别名>]
[-keystore <密钥库>] [-storepass <存储库口令>]
[-storetype <存储类型>] [-providername <名称>]
[-providerclass <提供方类名称> [-providerarg <参数>]] ...
[-providerpath <路径列表>]

-printcert　[-v] [-file <认证文件>]

-storepasswd [-v] [-new <新存储库口令>]
[-keystore <密钥库>] [-storepass <存储库口令>]
[-storetype <存储类型>] [-providername <名称>]
[-providerclass <提供方类名称> [-providerarg <参数>]] ...
[-providerpath <路径列表>]

(2) jarsigner 用法：
[选项] jar 文件别名
jarsigner -verify [选项] jar 文件

[-keystore <url>]	密钥库位置
[-storepass <口令>]	用于密钥库完整性的口令
[-storetype <类型>]	密钥库类型
[-keypass <口令>]	专用密钥的口令(如果不同)
[-sigfile <文件>]	.SF/.DSA 文件的名称
[-signedjar <文件>]	已签名的 JAR 文件的名称
[-digestalg <算法>]	摘要算法的名称
[-sigalg <算法>]	签名算法的名称
[-verify]	验证已签名的 JAR 文件
[-verbose]	签名/验证时输出详细信息
[-certs]	输出详细信息和验证时显示证书
[-tsa <url>]	时间戳机构的位置
[-tsacert <别名>]	时间戳机构的公共密钥证书
[-altsigner <类>]	替代的签名机制的类名
[-altsignerpath <路径列表>]	替代的签名机制的位置
[-internalsf]	在签名块内包含 .SF 文件
[-sectionsonly]	不计算整个清单的散列
[-protected]	密钥库已保护验证路径
[-providerName <名称>]	提供者名称
[-providerClass <类>]	加密服务提供者的名称
[-providerArg <参数>] ...	主类文件和构造函数参数

2. Eclipse的ADT生成

实际上，使用 Eclipse 可以更加直观、方便地生成签名文件，具体流程如下。

(1) 右击 Eclipse 项目名，依次选择 Android Tools | Export Signed Application Package 命令，如图 16-25 所示。

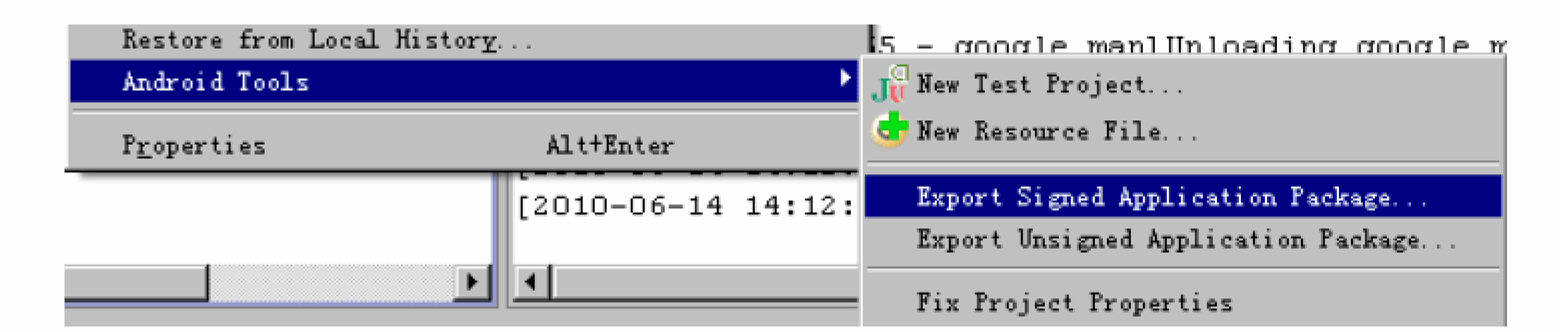


图 16-25　选择导出

(2) 在打开的对话框中选择要导出的项目，如图 16-26 所示。

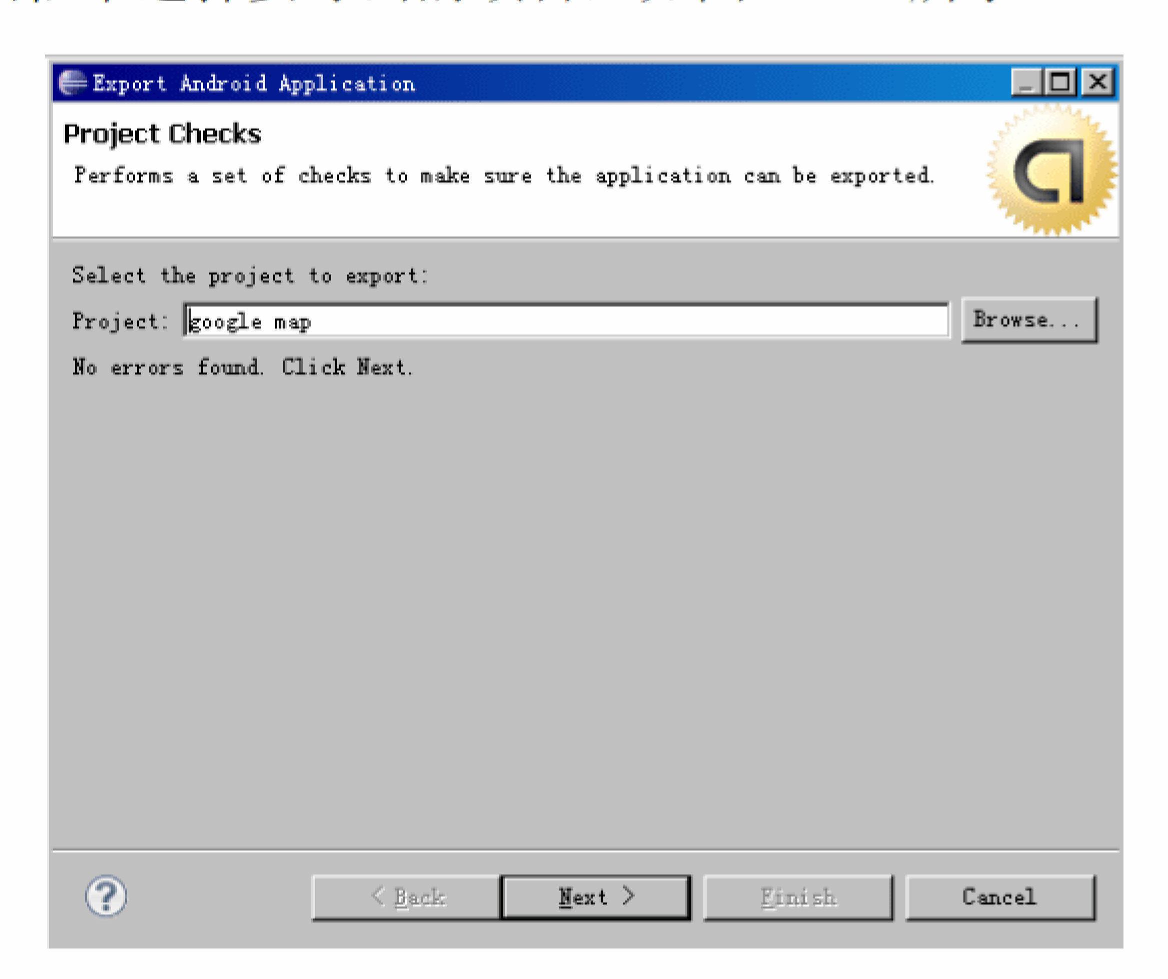


图 16-26　选择要导出的项目

(3) 单击 Next 按钮，在弹出的对话框中选中 Create new keystore 单选按钮，然后分别输入文件名和密码，如图 16-27 所示。

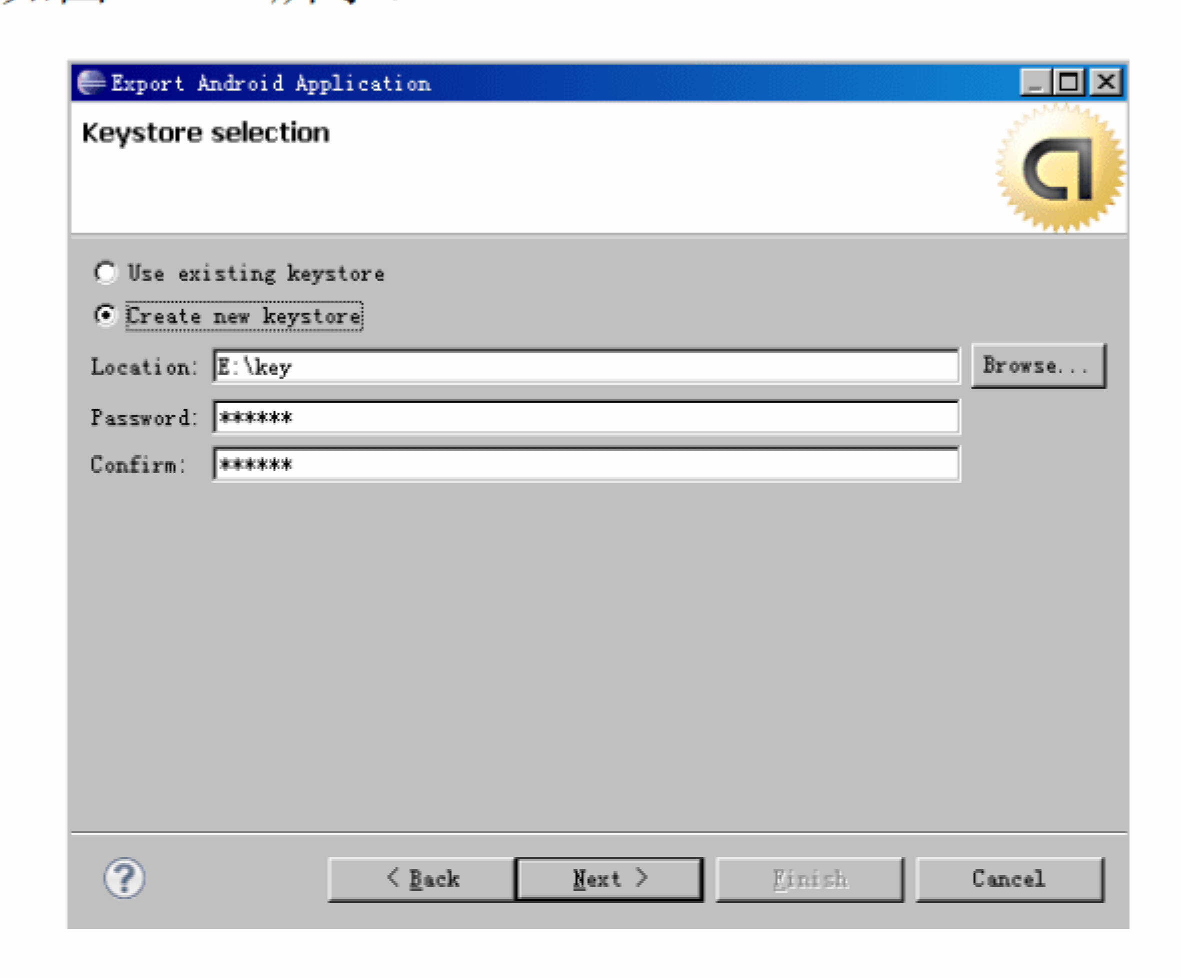


图 16-27　输入文件名和密码

(4) 单击 Next 按钮，在弹出的对话框中输入签名文件路径，如图 16-28 所示。

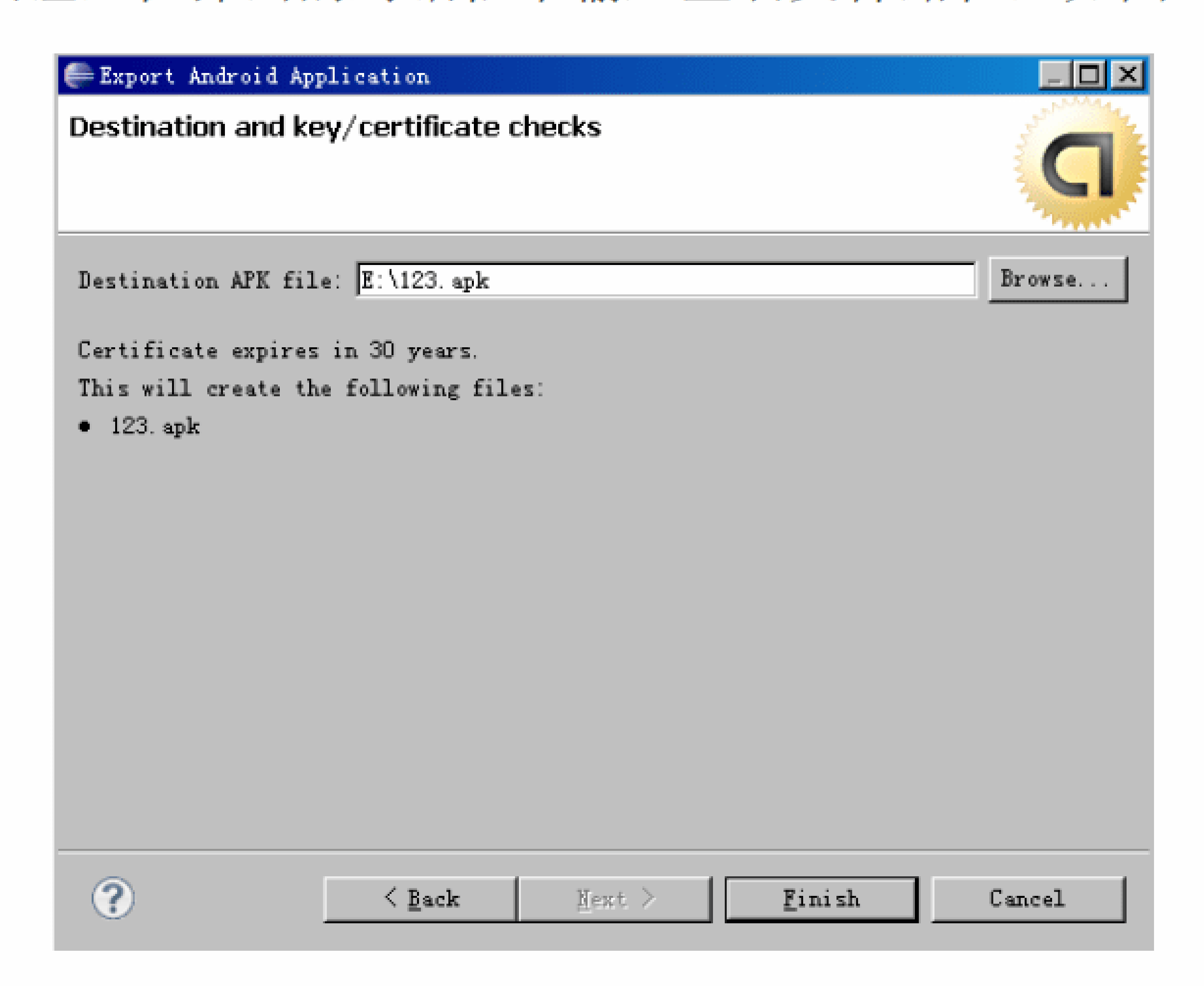


图 16-28　输入签名文件路径

(5) 单击 Next 按钮，在弹出的对话框中依次输入签名文件的相关信息，如图 16-29 所示。

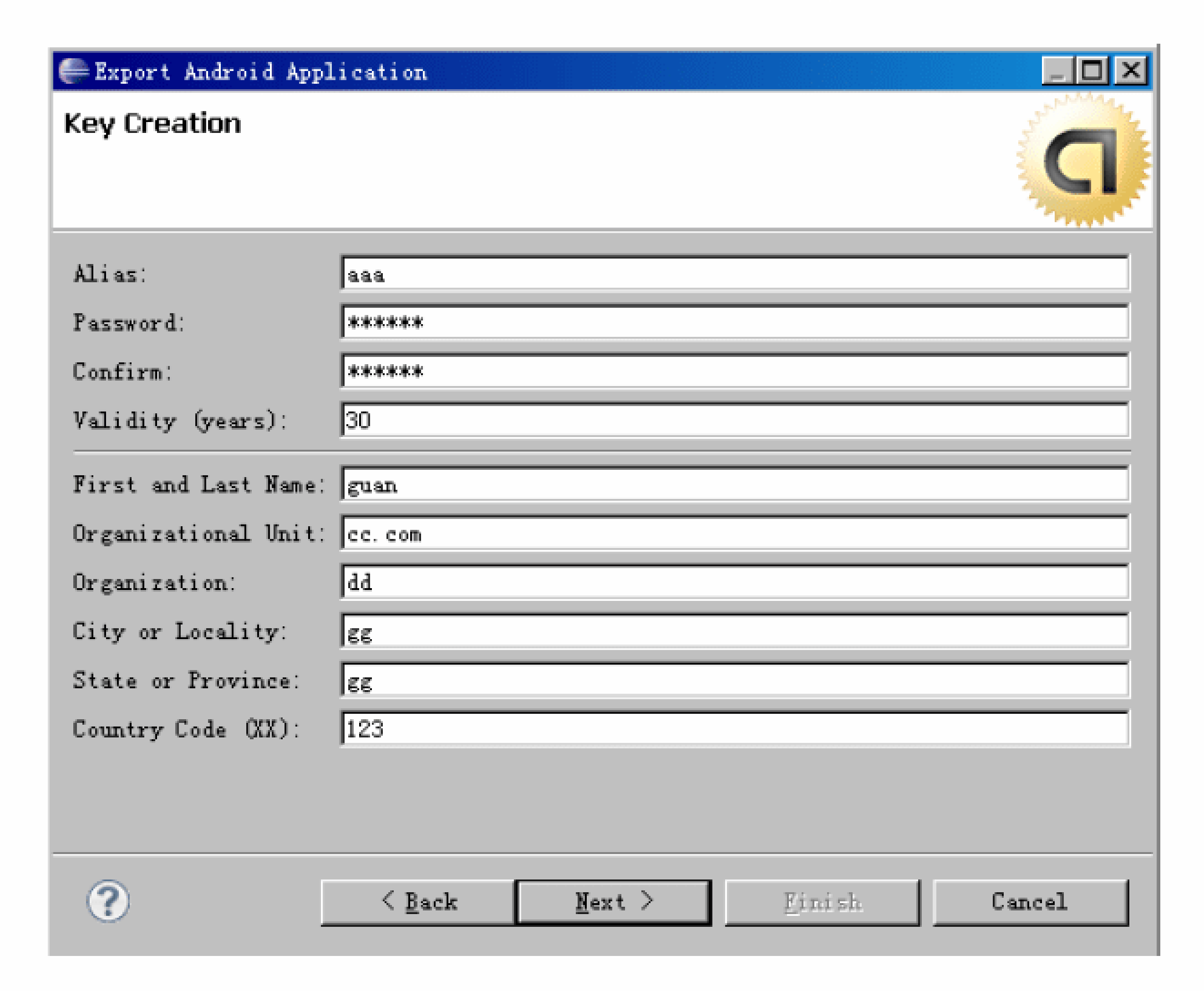


图 16-29　输入签名文件的相关信息

(6) 单击 Next 按钮后完成签名文件的创建。

16.5.3　使用签名文件

生成签名文件后，就可以使用它了。有以下两种使用方式。

(1) 假设生成的签名文件是 ChangeBackgroundWidget.apk，则最终生成 ChangeBackgroundWidget_signed.apk 为 Android 签名后的 APK 执行文件。

输入以下命令行：

```
jarsigner -verbose -keystore ChangeBackgroundWidget.keystore -signedjar
ChangeBackgroundWidget_signed.apk ChangeBackgroundWidget.apk
```

```
ChangeBackgroundWidget.keystore
```

上面命令中间不换行。

(2) 按 Enter 键，根据提示输入密钥库的口令短语(即密码)，详细信息如下：

输入密钥库的口令短语：

```
正在添加:  META-INF/MANIFEST.MF
正在添加:  META-INF/CHANGEBA.SF
正在添加: META-INF/CHANGEBA.RSA
正在签名: res/drawable/icon.png
正在签名: res/drawable/icon_audio.png
正在签名: res/drawable/icon_exit.png
正在签名: res/drawable/icon_folder.png
正在签名: res/drawable/icon_home.png
正在签名: res/drawable/icon_img.png
正在签名: res/drawable/icon_left.png
正在签名: res/drawable/icon_mantou.png
正在签名: res/drawable/icon_other.png
正在签名: res/drawable/icon_pause.png
正在签名: res/drawable/icon_play.png
正在签名: res/drawable/icon_return.png
正在签名: res/drawable/icon_right.png
正在签名: res/drawable/icon_set.png
正在签名: res/drawable/icon_text.png
正在签名: res/drawable/icon_xin.png
正在签名: res/layout/fileitem.xml
正在签名: res/layout/filelist.xml
正在签名: res/layout/main.xml
正在签名: res/layout/widget.xml
正在签名: res/xml/widget_info.xml
正在签名: AndroidManifest.xml
正在签名: resources.arsc
正在签名: classes.dex
```

通过上述过程处理后，即可将未签名文件 ChangeBackgroundWidget.apk 签名为 ChangeBackgroundWidget_signed.apk。

在上述方式中，读者可能会遇到以下问题。

问题一：jarsigner 无法打开 jar 文件 ChangeBackgroundWidget.apk。

解决方法：将要进行签名的 APK 放到对应的文件下，把要签名的 ChangeBackgroundWidget.apk 放到 JDK 的 bin 文件里。

问题二：jarsigner 无法对 jar 进行签名：

```
java.util.zip.ZipException: invalid entry comp
ressed size (expected 1598 but got 1622 bytes)
```

方法一：Android 开发网提示这些问题主要是由于资源文件造成的，对于 Android 开发来说应该检查 res 文件夹中的文件，逐个排查。这个问题可以通过升级系统的 JDK 和 JRE 版本来解决。

方法二：这是因为默认给 APK 做了 debug 签名，所以无法做新的签名，这时就必须

右击工程，在弹出的快捷菜单中选择 Android Tools | Export Unsigned Application Package 命令。

或者从 AndroidManifest.xml 的 Exporting 上也是一样的。

然后再基于这个导出的 unsigned apk 做签名，导出的时候最好将其目录选在你之前产生 keystore 的那个目录下，这样操作起来就方便了。

16.5.4 发布

发布的过程比较简单，来到 Market，登录个人中心，上传签名后的文件即可。具体操作流程在 Market 站点上有详细的介绍说明。为节省本书的篇幅，在此将不做详细介绍。